Chapitre 2:
Les opérateurs de base & les fonctions d'E/S standard

I. Introduction

Le langage C apporte quelques notions innovantes en matière d'opérateurs et de fonctions d'E/S.

En particulier, le langage C considère l'affectation comme un opérateur normal alors que les langages qui l'ont précédé (par exemple FORTRAN, ADA) la considèrent comme une opération privilégiée.

Dans ce chapitre on se limite aux principaux opérateurs du langage C et aux fonctions printf () et scanf () qui représentent les fonctions standard de la bibliothèque d'E/S «stdio.h».

II. Les opérateurs

- Les opérateurs sont les éléments du langage qui permettent de faire du calcul ou de définir des relations.
- Ils servent à combiner des variables et des constantes pour réaliser des expressions.

II. Les opérateurs

1) Les opérateurs arithmétiques

Le langage C connaît :

- deux opérateurs arithmétiques unaires : + et –
- cinq opérateurs binaires :
  - addition (+)
  - soustraction (-)
  - division (/)
  - multiplication (*)
  - modulo (%) (Reste de la division entière)

II. Les opérateurs

1) Les opérateurs arithmétiques

Exemple:

```
…
int a, b =125, c;
float x, y=2.75;
…
a= -b;
c=3*a+b%2;
…
```

II. Les opérateurs

2) Les opérateurs d'affectation

Le langage C permet de construire des opérateurs binaires d'affectation à partir des opérateurs binaires arithmétiques, des opérateurs de masquage et des opérateurs de décalage, en les faisant suivre d'un égal (=).

L'utilisation de ces opérateurs d'affectation composés permet la simplification des expressions.

opérateur	utilisation	équivalent
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Les opérateurs -- et ++ permettent de décrémenter et d'incrémenter des variables de type entier.
Dans une première approche, nous pouvons considérer que :

var-- ; est équivalent à var = var – 1 ;

➔ en partant d'une variable var ayant une valeur de 8 que cette variable contient la valeur 7 une fois l'expression calculée à l'exécution ;

var++ ; est équivalent à var = var + 1;

➔ en partant de var ayant une valeur de 8 que cette variable contient la valeur 9 à la fin de l'exécution de l'expression.

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Il est possible d'utiliser les opérateurs unaires d'incrémentation et de décrémentation derrière la variable (postfixé) ou devant celle-ci (préfixé).

Ce qui permet de post-incrémenter, de pré-incrémenter, de post-décrémenter ou de pré-décrémenter.

- Lorsque l'opérateur est préfixé, l'opération est appliquée avant que la valeur correspondant à l'opération ne soit calculée.
- Dans le cas où l'opération est post-fixée, la valeur de la variable avant l'opération est utilisée pour les autres calculs et ensuite l'opération est appliquée.

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Exemple 1 :

```
int i=0, j=0 ;
j = ++i ;
```

➔ C'est une pré-incrémentation de la variable i. Cela signifie incrémenter i de 1 puis mettre la valeur de i dans j. À la fin de cette opération i vaut 1 et j vaut 1.

➔ Donc on peut conclure que **j=++i** est équivalente aux deux instructions suivantes :

```
i = i+1 ;
j = i;
```

II. Les opérateurs

3) Les opérateurs d'incrémentation et de décrémentation

Exemples 2 :

```
int i=0, j=0;
j = i++ ;
```

➔ Cette instruction signifie mettre la valeur de i dans j puis incrémenter i de 1. En partant des mêmes valeurs, i valant 0 et j valant 0, à la fin de cette instruction i vaut 1 et j vaut 0.

➔ Donc on peut conclure que **j=i++** est équivalente aux deux instructions suivantes :

```
j=i ;
i = i+1 ;
```

II. Les opérateurs

4) Les opérateurs logiques (booléens)

Ce type d'opérateurs permet de vérifier si plusieurs conditions sont vraies:

Les deux opérateurs de tests (**&&** et **| |**) sont appelés respectivement le "et logique" et le "ou logique".

- Le "et logique" permet de décrire qu'une condition constituée de deux parties est satisfaite si et seulement si les deux parties sont satisfaites.
- Le "ou logique" permet de décrire qu'une condition constituée de deux parties est satisfaite si et seulement si l'une des deux parties est satisfaite.

II. Les opérateurs

4) Les opérateurs logiques (booléens)

- La négation logique (!) sert à inverser une condition en la faisant passer de vrai à faux et réciproquement.
- En langage C, une expression est fausse si la valeur qu'elle retourne est égale à 0, elle est vraie sinon. De plus, la norme spécifie que !0 vaut 1.

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

II. Les opérateurs

5) Les opérateurs de manipulation de bits

Les opérateurs de masquage et de décalages sont utilisés pour manipuler les bits des variables entières.

&	:	et logique	**~**	:	complément à 1
|	:	ou inclusif	**<<**	:	décalage à gauche
^	:	ou exclusif	**>>**	:	décalage à droite

```
int a = 6;   // 0110
int b = 3;   // 0011
int c = a | b; // c = 0111 = 7
```

```
int a = 6;   // 0110
int b = 3;   // 0011
int c = a ^ b; // c = 0101 = 5
```

```
int a = 6;   // 6 = 0110 en binaire
int b = 3;   // 3 = 0011
int c = a & b; // c = 0010 = 2
```

```
int a = 6;   //  0000...0110
int c = ~a;  //  1111...1001  (en décimal, dépend
de la taille de int → généralement -7)
```

II. Les opérateurs

5) Les opérateurs de manipulation de bits

Décalage à gauche (<<)

Décale les bits vers la gauche, en ajoutant des `0` à droite.

→ Multiplie par 2 pour chaque décalage.

```
int a = 3;       // 0011
int c = a << 1; // 0110 = 6
```

Décalage à droite (>>)

Décale les bits vers la droite.

→ Divise par 2 pour chaque décalage (en perdant les bits de droite).

```
int a = 6;       // 0110
int c = a >> 1; // 0011 = 3
```

II. Les opérateurs

6) Les opérateurs de comparaison

Les opérateurs de comparaison servent à réaliser des tests entre les valeurs de deux expressions.

Comme nous le verrons dans le chapitre 7 ces opérateurs sont surtout utilisés à l'intérieur des instructions de contrôle de flot (tests).

Les opérateurs de relation algébrique sont au nombre de six : (== < <= > >= !=).

II. Les opérateurs

6) Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
== A ne pas confondre avec le signe d'affectation (=)!!	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne 1 si X est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne 1 si X est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne 1 si X est inférieur ou égal à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne 1 si X est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne 1 si X est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne 1 si X est différent de 3, sinon 0

II. Les opérateurs

7) L'opérateur d'adressage

L'operateur d'adressage ou de référencement noté (&) permet de retourner l'adresse en mémoire (référence) de la variable dont le nom le suit.

Exemple

&var donne l'adresse en mémoire de la variable var.

II. Les opérateurs

8) L'opérateur de taille

L'opérateur **sizeof** donne la taille en octets de l'opérande selon la syntaxe:

sizeof expression

ou

sizeof (expression)

Exemple:

```
int a, b, c;
double x;
a= sizeof  (long) ;     /* a← 4 */
b= 3* sizeof x ;        /* b ← 24 */
c= sizeof "informatique"; /* c ←13 */
```

II. Les opérateurs

9) L'opérateur conditionnel (? :)

Cet opérateur est un opérateur **ternaire** qui met en jeu trois expressions de la façon suivante

expr1 ? expr2 : expr3

Il permet de construire une opération de test avec retour de valeur.

Si expr1 fournit une valeur non nulle,

alors la valeur de l'expr2 est fournie comme résultat

si non la valeur de l'expr3 est fournie comme résultat.

II. Les opérateurs

10) L'opérateur conditionnel (? :)

Exemple 1:

X= (a == b) ? c : d

Cette expression retourne la valeur de la variable c si la valeur de la variable a est égale à celle de la variable b. Dans le cas contraire, l'expression retourne la valeur de la variable d.

Exemple 2 :

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

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

III. Les Fonctions d'E/S standard

Les entrée-sorties en langage C ne sont pas prises en charge directement par le compilateur mais elles sont réalisées à travers des fonctions de bibliothèque.

Le compilateur ne peut pas faire de contrôle de cohérence dans les arguments passés à ces fonctions car ils sont de type variable.

Ceci explique l'attention toute particulière avec laquelle ces opérations doivent être programmées.

La bibliothèque standard d'E/S du langage C est la bibliothèque **<stdio.h>**

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ( )

Syntaxe :

printf ("String Format" [, argument1… argumentN]);

String format : chaîne de caractères avec des caractères de formatage si nécessaire.

Un argument : peut être une valeur ou une variable ou une expression ou un appel de fonction.
Le nombre des arguments après le String Format dépend du nombre de variables pour lesquels vous voulez afficher les valeurs.

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ( )

format	signification
%c	caractère
%s	chaîne de caractères
%d	nombre entier en décimal
%f	nombre réel sous la forme [-]mmm.nnnnnn
%e	nombre réel sous la forme mantisse/exposant [-]m.nnnnnne[+|-]xx
%g	nombre réel sous la forme la plus courte entre les types %e et %f
%o	nombre entier en octal
%p	pointeur ou adresse de la valeur numérique
%u	nombre entier non signé en décimal
%x	nombre entier en hexadécimal
%X	nombre entier en hexadécimal ; lettres affichées en majuscules

III. Les Fonctions d'E/S standard

1) La fonction d'affichage : printf ( )

Exemple:

```
int x=8 ;
float y=5.6 ;
char c= 'b' ;

printf(" la valeure de la variable  c = %c", c) ;
printf(" l'adresse de la variable c = %x", &c) ;
printf("x= %d et y=%f", x, y) ;
```

III. Les Fonctions d'E/S standard

1) La fonction de saisie : scanf ( )

C'est une fonction implémentée dans la bibliothèque standard **stdio.h**, elle permet la saisie des variables : elle modifie leurs contenus.

Syntaxe :

```
scanf ("Format1 […Formatn]", &argument1[, … &argumentN]);
```

Format doit être un format du tableau précédent (%d, %f…)
Argument ne peut être qu'une variable dont la valeur va être saisie par l'utilisateur.
L'utilisation de l'opérateur & est obligatoire sauf pour la saisie des variables de type pointeur.

III. Les Fonctions d'E/S standard

1) La fonction de saisie : scanf ( )

Exemple 1:

```
int x;
float y;
scanf ("%d %f" , &x, &y);
```

Exemple 2:

```
#include <stdio.h>
void main()
{
        int i;
        printf("entrez un entier sous forme hexadécimale i =");
        scanf("%x", &i);
        printf("i = %d\n", i);
}
/*Si la valeur 1a est saisie, alors le programme affiche i = 26*/
```

IV. Exercices d'application

Exercice 1:

Ecrire un programme qui permute et affiche les valeurs de trois variables A, B, C de type entier qui sont entrées au clavier :

A ==> B , B ==> C , C ==> A

Exercice 2:

Ecrire un programme qui affiche la résistance équivalente à trois résistances R1, R2, R3 (type **double**),

- si les résistances sont branchées en série:

$$R_{sér} = R1+R2+R3$$

- si les résistances sont branchées en parallèle:

$$R_{par} = (R1.R2.R3)/(R1.R2+R1.R3+R3.R2)$$

IV. Exercices d'application

Exercice 3:

Définir deux entiers i et j initialisés avec les valeurs 10 et 3 respectivement.

Faire écrire les résultats de :

i + j puis i – j puis i * j puis i / j puis i % j

Affecter les valeurs 0x0FF et 0xF0F respectivement à i et j.

Faire écrire en hexadécimal les résultats de :

i & j puis i | j puis i ^ j puis i << 2 puis j >> 2

Exercice 4:

En utilisant 4 entiers i, j, k et l, avec k initialisé à 12 et l à 8, écrire le programme qui :

- lit les valeurs de i et j
- écrit la valeur de k si i est nul ;
- écrit la valeur de i + l si i est non nul et j est nul
- écrit la valeur de i + j dans les autres cas.

IV. Exercices d'application

Solution de l'exercice 1:

```
#include <stdio.h>
void main()
{
        int A, B, C, AIDE;
        printf("Introduisez trois nombres (A, B, C) : ");
        scanf("%i %i %i", &A, &B, &C);
                /* Affichage à l'aide de tabulations */
        printf("A = %i\tB = %i\tC = %i\n", A, B, C);
        AIDE=A; A=C; C=B; B=AIDE;
        printf("A = %i\tB = %i\tC = %i\n", A, B, C);
        return 0;
}
```

IV. Exercices d'application

Solution de l'exercice 2:

```
#include <stdio.h>
void main()
{
        double R1, R2, R3, RRES; printf("Introduisez les
        valeurs pour R1, R2 et R3 : ");
        scanf("%lf %lf %lf", &R1, &R2, &R3);
        RRES=R1+R2+R3;
        printf("Resistance resultante serielle : %f\n",
        RRES);
        RRES=(R1*R2*R3)/(R1*R2+R1*R3+R2*R3);
        printf("Resistance resultante parallele : %f\n",
        RRES); return 0;
}
```

IV. Exercices d'application

Solution de l'exercice 4:

```
#include <stdio.h>
 void main ( )
{
        /* definition de 4 entiers */
        int i, j, k = 12, l = 8;
        /* lecture des valeurs de i et de j */
        printf("\n Entrer la valeur de i :");
        scanf("%d", &i);
        printf("\n Entrer la valeur de j :");
        scanf("%d", &j);

        /* ecriture du resultat selon les valeurs de i et de j*/
         printf("\n resultat : %d\n", (!i ? k : (!j ? i + l : i + j)));
}
```