

Chapitre II

CONCEPTS DE BASE

Objectifs

Général

⇒ Faire acquérir à l'apprenant les notions de base relative à la programmation en C.

Spécifiques

- ❑ Connaître les opérateurs du langage C ;
 - ❑ Connaître les principaux types de données du langage C ;
 - ❑ Se familiariser avec les fonctions assurant l'échange d'information entre la mémoire centrale et les périphériques standard.
-

1. Types de base, opérateurs et expressions

Les *variables* et les *constantes* sont les données principales qui peuvent être manipulées par un programme. Les *déclarations* introduisent les variables qui sont utilisées, fixent leur type et parfois aussi leur valeur de départ.

Les *opérateurs* contrôlent les actions que subissent les valeurs des données. Pour produire de nouvelles valeurs, les variables et les constantes peuvent être combinées à l'aide des opérateurs dans des *expressions*.

Le *type* d'une donnée détermine l'ensemble des valeurs admissibles, le nombre d'octets à réserver en mémoire et l'ensemble des opérateurs qui peuvent y être appliqués.

1.1. Les types simples

→ Ensembles de nombres et leur représentation

En mathématiques, nous distinguons divers ensembles de nombres:

- L'ensemble des entiers naturels $\mathbb{N}$,
- L'ensemble des entiers relatifs $\mathbb{Z}$,
- L'ensemble des rationnels $\mathbb{Q}$,
- L'ensemble des réels $\mathbb{R}$.

Un ordinateur ne peut traiter aisément que des nombres entiers d'une taille limitée. Il utilise le système binaire pour calculer et sauvegarder ces nombres.

→ Les charges du programmeur

Un programmeur utilisant C doit être capable de :

- Choisir un type numérique approprié à un problème donné,
 - Choisir un type approprié pour la représentation sur l'écran,
 - Prévoir le type résultant d'une opération entre différents types numériques,
 - Prévoir et optimiser la précision des résultats intermédiaires au cours d'un calcul complexe.
-

Avant de pouvoir utiliser une variable, nous devons nous intéresser à deux caractéristiques de son type numérique :

- Le domaine des valeurs admissibles,
- Le nombre d'octets qui est réservé pour une variable.

Le tableau suivant résume les caractéristiques des types numériques entiers de C :

<i>Définition</i>	<i>Description</i>	<i>Domaine min</i>	<i>Domaine max</i>	<i>Nombre d'octets</i>
char	caractère	-128	127	1
short	entier court	-32768	32767	2
int	entier standard	-32768	32767	2
long	entier long	-2147483648	2147483647	4
float	Réel	$3.4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	4

1.2. Conversion de type de données

On appelle *conversion de type de données* le fait de modifier le type d'une donnée en une autre. Il peut arriver par exemple que l'on veuille travailler sur un type de variable, puis l'utiliser sous un autre type.

Imaginons que l'on travaille par exemple sur une variable en virgule flottante (type *float*), il se peut que l'on veuille "supprimer les chiffres après la virgule", c'est-à-dire convertir un *float* en *int*. Cette opération peut être réalisée de deux manières :

- **Conversion implicite**

Elle consiste en une modification du type de donnée effectuée automatiquement par le compilateur. Cela signifie que lorsque l'on va stocker un type de donnée dans une variable déclarée avec un autre type, le compilateur ne retournera pas d'erreur mais effectuera une conversion *implicite* de la donnée avant de l'affecter à la variable.

```
int x;
x = 8.324;
```

→ *x* contiendra après affectation la valeur 8.

Exercice 2.1

Soient les déclarations :

```
long  A = 15;
char  B = 'A';          /* code ASCII : 65 */
short C = 10;
```

Quels sont le type et la valeur de chacune des expressions :

- (1) *C + 3*
- (2) *B + 1*
- (3) *C + B*
- (4) *3 * C + 2 * B*
- (5) *2 * B + (A + 10) / C*
- (6) *2 * B + (A + 10.0) / C*

Solution

	Expression	Type	Valeur
(1)	$C + 3$	int	13
(2)	$B + 1$	int	66
(3)	$C + C$	int	75
(4)	$3 * C + 2 * B$	int	160
(5)	$2 * B + (A + 10) / C$	long	132
(6)	$2 * B + (A + 10.0) / C$	double	132.5

▪ Conversion explicite

Elle consiste en une modification du type de donnée forcée. Cela signifie que l'opérateur est tout simplement le type de donnée, dans lequel on désire convertir une variable, entre des parenthèses précédant la variable.

```
int x;
x = (int)8.324;
```

→ x contiendra après affectation la valeur 8

1.3. Les variables du langage C

1.3.1. La déclaration de variables

Pour pouvoir utiliser une variable, il faut la définir, c'est-à-dire lui donner un nom, mais surtout un type de donnée à stocker afin qu'un espace mémoire conforme au type de donnée qu'elle contient lui soit réservé.

Une variable se déclare de la façon suivante :

```
type Nom_de_la_variable;
```

ou bien s'il y a plusieurs variables du même type :

```
type Nom_de_la_variable1, Nom_de_la_variable2, ...;
```

Exercice 2.2

Traduisez les déclarations suivantes en C, sachant que vous travaillerez dans les ensembles de nombres indiqués. Choisissez les types les plus économiques, sans perdre en précision.

- 1) entier COMPTEUR {0 ,..., 300}
- 2) entier X,Y {-120 ,..., 100}
- 3) entier MESURE {-10 ,..., 10⁴}
- 4) réel SURFACE1 {0.5 ,..., 150075}
- 5) réel SURFACE2 {-12 ,..., 1500750.5}
- 6) entier N1 {0 ,..., 2¹⁰}
- 7) entier N2 {-4⁷ ,..., 4⁷}
- 8) entier N4 {0 ,..., 32⁶}
- 9) entier N4 {-128⁰ ,..., 128⁵}
- 10) booléen TROUVE {vrai, faux}

Solution

- | | |
|------------------|--------------|
| 1) int COMPTEUR; | 2) char X,Y; |
|------------------|--------------|

3) int MESURE;	7) int N2;
4) float SURFACE1;	8) long N3;
5) double SURFACE2;	9) double N4;
6) int N1;	10) int TROUVE ;

1.3.2. Affectation d'une donnée à une variable

Pour stocker une donnée dans une variable que l'on a initialisée, il faut faire une affectation, c'est-à-dire préciser la donnée qui va être stockée à l'emplacement mémoire qui a été réservé lors de l'initialisation.

Pour cela on utilise l'opérateur d'affectation "=" :

```
Nom_de_la_variable = donnée;
```

Pour stocker le caractère B dans la variable que l'on a appelée *Caractere*, il faudra écrire:

```
Caractere = 'B' ;
```

Ce qui signifie *stocker la valeur ASCII de "B" dans la variable nommée "Caractere"*. Il est bien évident qu'il faut avoir préalablement déclaré la variable en lui affectant le type *char*:

```
char Caractere;
```

1.3.3. Initialisation d'une variable

On peut affecter une valeur initiale à la variable lors de sa déclaration, on parle alors d'initialisation:

```
type Nom_de_la_variable = donnee;
```

Par exemple: `float Toto = 125.36;`

1.3.4. Définition de constantes

Une constante est une variable dont la valeur est inchangeable lors de l'exécution d'un programme. En langage C, les constantes sont définies grâce à la directive du préprocesseur *#define*, qui permet de remplacer toutes les occurrences du mot qui le suit par la valeur immédiatement derrière elle. Par exemple la directive:

```
#define Pi 3.1415927
```

Il est ainsi préférable d'utiliser le mot clef *const*, qui permet de déclarer des constantes typées :

```
const int dix = 10;
```

1.4. Les opérateurs du langage C

1.4.1. Type d'opération

Les opérateurs sont des symboles qui permettent de manipuler des variables, c-à-d effectuer des opérations, les évaluer, etc. On distingue plusieurs types d'opérateurs:

⇒ Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable.

Opérateur	Dénomination	Effet
+	opérateur d'addition	Ajoute deux valeurs
-	opérateur de soustraction	Soustrait deux valeurs

*	opérateur de multiplication	Multiplie deux valeurs
/	opérateur de division	Divise deux valeurs
=	opérateur d'affectation	Affecte une valeur à une variable

⇒ Les opérateurs d'incrémentement

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Opérateur	Dénomination	Effet	Syntaxe
++	Incrémentement	Augmente d'une unité la variable	x++
--	Décrémentement	Diminue d'une unité la variable	x--

⇒ Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles que ajouter une valeur dans une variable et stocker le résultat dans la variable.

Opérateur	Effet
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
*=	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable
%=	Modulo

⇒ Les opérateurs de comparaison

Opérateur	Dénomination	Effet
==	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur

⇒ Les opérateurs logiques (booléens)

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

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	(!condition)

1.4.2. Priorité d'un opérateur

On dit alors que la multiplication *à la priorité* sur l'addition et que la multiplication et l'addition ont la priorité sur l'affectation.

Si nous voulons forcer l'ordinateur à commencer par un opérateur avec une priorité plus faible, nous devons (comme en mathématiques) entourer le terme en question par des *parenthèses*.

Exemple

Dans l'instruction: $X = 2*(A+3)*B+4*C$.

L'ordinateur évalue d'abord l'expression entre parenthèses, ensuite les multiplications, ensuite l'addition et enfin l'affectation.

Entre les opérateurs que nous connaissons jusqu'ici, nous pouvons distinguer les classes de priorités suivantes:

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

Exercice 2.3

Evaluer les expressions suivantes en supposant

a=20 b=5 c=-10 d=2 x=12 y=15

Notez chaque fois la valeur rendue comme résultat de l'expression et les valeurs des variables dont le contenu a changé.

- (1) $(5*X)+2*((3*B)+4)$
- (2) $(5*(X+2)*3)*(B+4)$
- (3) $A == (B=5)$
- (4) $A += (X+5)$
- (5) $A != (C * (-D))$
- (6) $A *= C+(X-D)$
- (7) $A \% = D++$
- (8) $A \% = ++D$
- (9) $(X++)*(A+C)$
- (10) $A = X*(B<C)+Y*!(B<C)$
- (11) $!(X-D+C) || D$
- (12) $A \&\&B || !0 \&\&C \&\&!D$
- (13) $((A \&\&B) || (!0 \&\&C)) \&\&!D$
- (14) $((A \&\&B) || !0) \&\&(C \&\&(!D))$

Solution

(1)	$(5*X)+2*((3*B)+4)$	→ 98	/
(2)	$(5*(X+2)*3)*(B+4)$	→ 1890	/
(3)	$A == (B=5)$	→ 0	B=5
(4)	$A += (X+5)$	→ 37	A=37

(5)	A != (C * (-D))	→ 0	C=20
(6)	A *= C+(X-D)	→ 0	A=0
(7)	A %= D++	→ 0	D=3 A=0
(8)	A %= ++D	→ 2	D=3 A=2
(9)	(X++)*(A+C)	→ 120	X=13
(10)	A = X*(B<C)+Y*!(B<C)	→ 0+15 = 15	A=15
(11)	!(X-D+C) D	→ !0 1 = 1	/
(12)	A&&B !0&&C&&!D	→ 1 1&&1&&0 = 1	/
(13)	((A&&B) (!0&&C))&&!D	→ (1 1)&&0 = 0	/
(14)	((A&&B) !0)&&(C&&(!D))	→ (1 1)&&(1&&0) = 0	/

1.5. Les fonctions arithmétiques standard

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

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

La liste des fonctions ne cite que les fonctions les plus courantes. Pour la liste complète et les constantes prédéfinies voir *<math.h>*.

1.5.1. Type des données

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

1.5.2. Fonctions arithmétiques

COMMANDE C	EXPLICATION	LANG. ALGORITHMIQUE
exp(X)	fonction exponentielle	e^X
log(X)	Logarithme naturel	$\ln(X)$, $X>0$
log10(X)	Logarithme à base 10	$\log_{10}(X)$, $X>0$
pow(X,Y)	X exposant Y	X^Y
sqrt(X)	racine carrée de X	pour $X>0$
fabs(X)	valeur absolue de X	$ X $
floor(X)	arrondir en moins	int(X)
ceil(X)	arrondir en plus	
fmod(X,Y)	reste rationnel de X/Y (même signe que X)	pour X différent de 0
sin(X) cos(X) tan(X)	sinus, cosinus, tangente de X	
asin(X) acos(X) atan(X)	arcsin(X), arccos(X), arctan(X)	

Exercice 2.4

Essayez le programme suivant

```
#include <stdio.h>
main( )
{
    double A;
    double B;
    double RES;
    /* Saisie de A et B */
    printf("Introduire la valeur pour A : ");
    scanf("%lf", &A);
    printf("Introduire la valeur pour B : ");
    scanf("%lf", &B);
    /* Calcul */
    RES = A*A;
    /* Affichage du résultat */
```

```

    printf("Le carré de A est %f \n", RES);
    /* Calcul */
    RES = B*B;
    /* Affichage du résultat */
    printf("Le carré de B est %f \n", RES);
    return 0;
}

```

Modifiez-le de façon à ce qu'il affiche:

- ✓ A^B ,
- ✓ l'hypoténuse d'un triangle rectangle de côtés A et B,
- ✓ la tangente de A en n'utilisant que les fonctions sin et cos,
- ✓ la valeur arrondie (en moins) de A/B,
- ✓ la valeur arrondie (en moins) à trois positions derrière la virgule de A/B.

Solution

```

#include <stdio.h>
#include <math.h>
main( )
{
    double A; double B; double RES;
    /* Saisie de A et B */
    printf("Introduire la valeur pour A : ");
    scanf("%lf", &A);
    printf("Introduire la valeur pour B : ");
    scanf("%lf", &B);
    /* Calcul */
    RES = pow(A,B);
    /* Affichage du résultat */
    printf("A exposant B est %f \n", RES);
    /* Calcul */
    RES = sqrt(pow(A,2)+pow(B,2));
    /* Affichage du résultat */
    printf("L'hypoténuse du triangle rectangle est %f \n", RES);
    /* Calcul */
    RES = sin(A)/cos(A);
    /* Affichage du résultat */
    printf("La tangente de A est %f \n", RES);
    /* Calcul */
    RES = floor(A/B);
    /* Affichage du résultat */
    printf("La valeur arrondie en moins de A/B est %f \n", RES);
    /* Calcul */
    RES = floor(1000*(A/B))/1000;
    /* Affichage du résultat */
    printf("La valeur A/B arrondie à trois décimales: %f \n", RES);
    return 0;
}

```

2. Lire et écrire des données

La bibliothèque standard `<stdio>` contient un ensemble de fonctions qui assurent la communication de la machine avec le monde extérieur. Dans cette section, nous allons en discuter les plus importantes.

- ✓ **printf()** : écriture de données,

- ✓ **scanf()** : lecture de données,
- ✓ **putchar()** : écriture d'un caractère,
- ✓ **getchar()** : lecture d'un caractère.

2.1. Écriture de données

La fonction printf est utilisée pour transférer du texte, des valeurs de variables ou des résultats d'expressions vers le fichier de sortie standard.

2.1.1. Syntaxe

Écriture en langage algorithmique

écrire (<Expression1>, <Expression2>, ...)

Écriture formatée en C

printf("<format>", <Expr1>, <Expr2>, ...)

Exemple 1

La suite d'instructions:

```
int A = 1234;
int B = 567;
printf("%d fois %d est %ld\n", A, B, (long)A*B);
```

va afficher sur l'écran: *1234 fois 567 est 699678*

Exemple 2

La suite d'instructions:

```
char B = 'A';
printf("Le caractère %c a le code %i !\n", B, B);
```

va afficher sur l'écran:

Le caractère A a le code 65 !

2.1.2. Spécificateurs de format pour printf

<i>Symbole</i>	<i>Type</i>	<i>Impression comme</i>
%d ou %i	int	entier relatif
%u	int	entier naturel (unsigned)
%o	int	entier exprimé en octal
%x	int	entier exprimé en hexadécimal
%c	int	caractère
%f	float	rationnel en notation décimale
%e	float	rationnel en notation scientifique

2.1.3. Arguments du type long

Les spécificateurs %d, %i, %u, %o, %x peuvent seulement représenter des valeurs du type int. Une valeur trop grande pour être codée dans deux octets est coupée sans avertissement si nous utilisons %d. Pour pouvoir traiter correctement les arguments du type long, il faut utiliser les spécificateurs %ld, %li, %lu, %lo, %lx.

Exercice 2.5

Soit le programme C suivant

```
#include <stdio.h>
main( )
{
    int N=10, P=5, Q=10, R;
    char C='S';

    N = 5;
    P = 2;
    Q = N++ > P || P++ != 3;
    printf ("C : N=%d P=%d Q=%d\n", N, P, Q);

    N = 5;
    P = 2;
    Q = N++ < P || P++ != 3;
    printf ("D : N=%d P=%d Q=%d\n", N, P, Q);

    N = 5;
    P = 2;
    Q = ++N == 3 && ++P == 3;
    printf ("E : N=%d P=%d Q=%d\n", N, P, Q);

    N=5;
    P=2;
    Q = ++N == 6 && ++P == 3;
    printf ("F : N=%d P=%d Q=%d\n", N, P, Q);

    N=C;
    printf ("G : %c %c\n", C, N);
    printf ("H : %d %d\n", C, N);
    printf ("I : %x %x\n", C, N);
    return 0;
}
```

Sans utiliser l'ordinateur, trouvez et notez les résultats du programme ci-dessus.

Solution

Voici les résultats du programme :

```
C : n=6 p=2 q=1
D : n=6 p=3 q=1
E : n=6 p=2 q=0
F : n=6 p=3 q=1
G : S S
H : 83 83
I : 53 53
```

2.2. Lecture de données

La fonction `scanf` est la fonction symétrique à `printf`; elle nous offre pratiquement les mêmes conversions que `printf`, mais en sens inverse.

2.2.1. Syntaxe

Lecture formatée en langage algorithmique

lire (<Nom Variable1>, <Nom Variable2>, ...)

Lecture formatée en C

```
scanf("<format>", <AdrVar1>, <AdrVar2>, ...)
```

L'adresse d'une variable est indiquée par le nom de la variable précédé du signe &.

Exemple

La suite d'instructions:

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

lit trois entiers relatifs, séparés par des espaces, tabulations ou interlignes. Les valeurs sont attribuées respectivement aux trois variables JOUR, MOIS et ANNEE.

2.2.2. Spécificateurs de format pour *scanf*

<i>Symbole</i>	<i>Type</i>	<i>Impression comme</i>
%d ou %i	int	entier relatif
%u	int	entier naturel (unsigned)
%o	int	entier exprimé en octal
%x	int	entier exprimé en hexadécimal
%c	int	caractère
%f	float	rationnel en notation décimale
%e	float	rationnel en notation scientifique

Exercice 2.6

Ecrire un programme qui lit la date du clavier et écrit les données ainsi que le nombre de données correctement reçues sur l'écran.

Exemple

Introduisez la date (jour mois année): 11 11 1991

jour : 11 mois : 11 année : 1991

Solution

```
#include <stdio.h>
main( )
{
    int JOUR, MOIS, ANNEE;
    printf("Introduisez la date (JOUR, MOIS, ANNEE) : ");
    scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
    printf("jour : %i\nmois : %i\nannée : %i\n", JOUR, MOIS, ANNEE);
    return 0;
}
```

2.3. Écriture d'un caractère

La commande **putchar('a');** transfère le caractère *a* vers le fichier standard de sortie *stdout*. Les arguments de la fonction **putchar** sont ou bien des caractères (c.-à-d. des nombres entiers entre 0 et 255) ou bien le symbole **EOF** (*End Of File*).

Type de l'argument

Pour ne pas être confondue avec un caractère, la constante **EOF** doit nécessairement avoir une valeur qui sort du domaine des caractères (en général **EOF** a la valeur -1). Ainsi, les

arguments de **putchar** sont par définition du type **int** et toutes les valeurs traitées par **putchar** (même celles du type **char**) sont d'abord converties en **int**.

Exemples

```
char A = 225;
char B = '\a';
int C = '\a';
putchar('x'); /* afficher la lettre x */
putchar('?'); /* afficher le symbole ? */
putchar('\n'); /* retour à la ligne */
putchar(65);
/* afficher le symbole avec le code 65 (ASCII: 'A') */
putchar(A);
/* afficher la lettre avec le code 225 (ASCII: 'ß') */
putchar(B); /* beep sonore */
putchar(C); /* beep sonore */
putchar EOF; /* marquer la fin du fichier */
```

2.4. Lecture d'un caractère

Une fonction plus souvent utilisée que **putchar** est la fonction **getchar**, qui lit le prochain caractère du fichier d'entrée standard *stdin*.

Type du résultat

Les valeurs retournées par **getchar** sont ou bien des caractères (0 - 255) ou bien le symbole **EOF**. Comme la valeur du symbole **EOF** sort du domaine des caractères, le type résultat de **getchar** est **int**. En général, **getchar** est utilisé dans une affectation:

```
int C;
C = getchar();
```

getchar lit les données de la zone tampon de *stdin* et fournit les données seulement après confirmation par 'Enter'. La bibliothèque *<conio>* contient une fonction du nom **getch** qui fournit immédiatement le prochain caractère entré au clavier.

La fonction **getch** n'est pas compatible avec ANSI-C et elle peut seulement être utilisée sous MS-DOS.

Exercice 2.7.

Ecrire un programme qui lit un caractère au clavier et affiche le caractère ainsi que son code numérique en employant:

- a) **getchar** et **printf**,
- b) **getch** et **printf**.

Solutions

a) En employant **getchar** et **printf**:

```
#include <stdio.h>
main( )
{
    int C;
    printf("Introduire un caractère suivi de 'Enter'\n");
    C = getchar();
    printf("Le caractère %c a le code ASCII %d\n", C, C);
}
```

```
    return 0;
}
```

b) En employant **getch** et **printf**

```
#include <stdio.h>
#include <conio.h>
main( )
{
    int C;
    printf("Introduire un caractère \n");
    C = getch();
    printf("Le caractère %c a le code ASCII %d\n", C, C);
    return 0;
}
```

3. Exercices d'application

Exercice 2.8

Ecrire un programme C qui calcule et affiche l'aire d'un triangle dont il faut entrer les longueurs des trois côtés A, B et C.

Utiliser la formule :

$$S^2 = P * (P - A) * (P - B) * (P - C)$$

Où A, B, C sont les longueurs des trois côtés (type int) et P le demi-périmètre du triangle.

Exercice 2.9

Ecrire un programme C qui: étant donné une longueur exprimée par un nombre entier de centimètres, calcule une longueur équivalente exprimée en pieds et pouces, puis affiche le résultat. On rappelle qu'un pied est égal à 12 pouces et qu'un pouce vaut 2.54 cm.

Exercice 2.10

Un ouvrier est payé 3,397 DT pour les heures de base et 4,322 DT pour les heures supplémentaires. Il subit une retenue de base de 2,2% sur son salaire brut (le salaire brut est le salaire obtenu en tenant compte des heures de base, des heures supplémentaires et de la retenue de base). Ecrire un programme C qui, connaissant le nombre d'heures de base et le nombre d'heures supplémentaires permet de calculer le salaire final de l'ouvrier.

Exercice 2.11

Ecrire un programme C qui lit un entier positif inférieur à 999 (composé de 3 chiffres au maximum) et affiche le nombre de centaine, de dizaine et d'unité.

Exercice 2.12

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

- Si les résistances sont branchées en série:

$$R_{\text{serie}} = R_1 + R_2 + R_3$$

- Si les résistances sont branchées en parallèle:

$$R_{\text{parallele}} = \frac{R_1 \cdot R_2 \cdot R_3}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3}$$

Exercice 2.13

Ecrire un programme C qui permet de :

- Lire une température en Celsius,
 - Effectuer la conversion en Fahrenheit en utilisant la formule : $T_f = \left(\frac{9}{5}\right) * (T + 32)$,
 - Afficher le résultat de la conversion sur l'écran.
-