Université Moulay Ismail
Faculté des Sciences et Techniques
Département d'Informatique
Errachidia

Année universitaire : 2020/2021
Filière MIP (S1) : Module I111
Prof Yousef FARHAOUI

Chapitre II (suite): les éléments de base d'un algorithme et leur représentation en langage C

(De l'algorithmique au langage C)

Structure d'un programme C

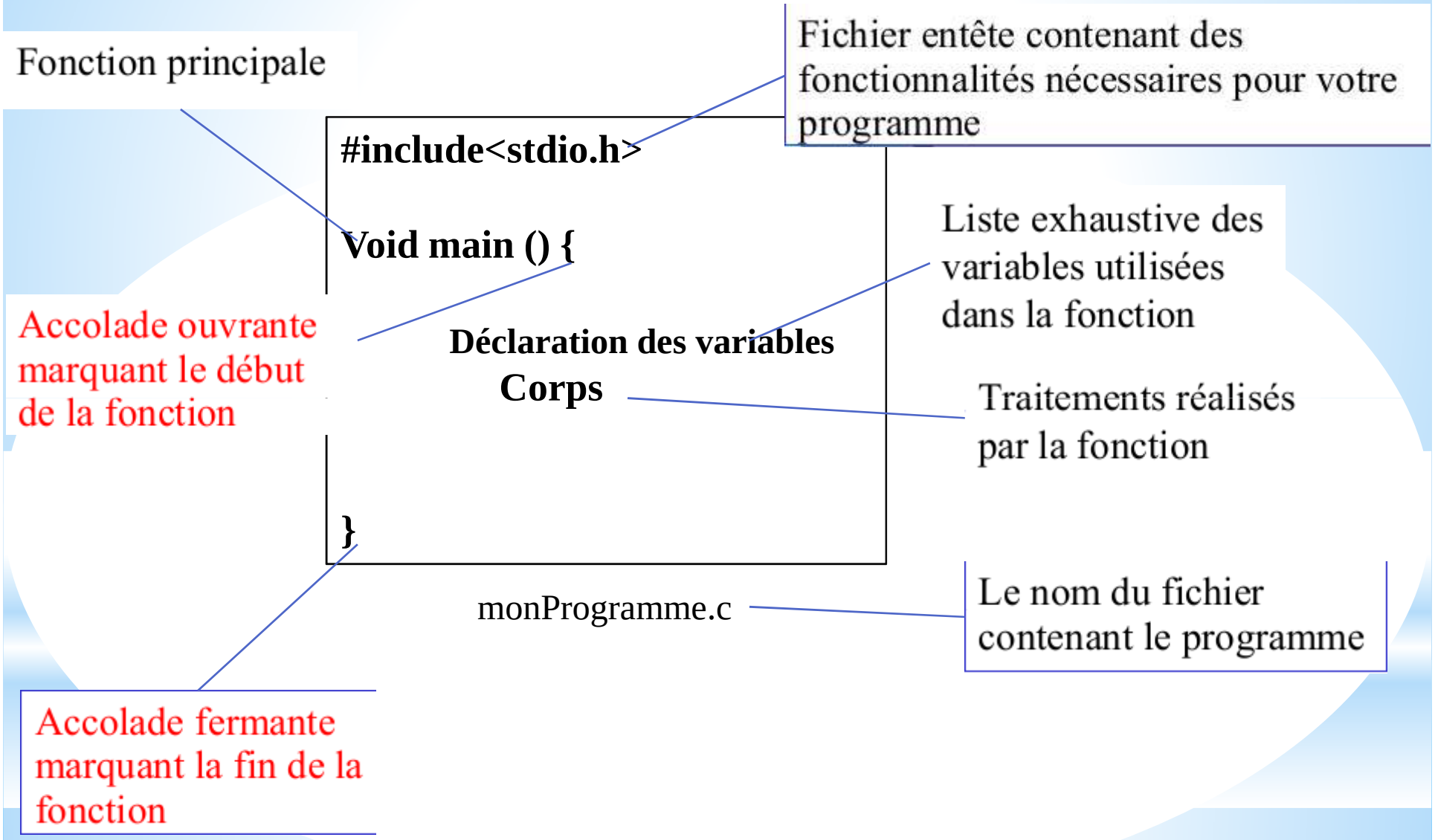
La structure d'un programme C est la suivante :

```
#include<stdio.h>
void main () {
        Déclaration des variables
        Corps
}
```

monProgramme.c

Votre programme doit Obligatoirement contenir une fonction principale « **main ( )** », qui est exécutée lorsque le programme est lancé

Structure d'un programme C



Structure d'un programme C

Mon premier programme : Bonjour tout le monde

```
#include <stdio.h>
void main()
{
printf(" Bonjour tout le monde ") ;

}
```

On sauvegarde ce programme dans un fichier qui se nomme **programme1.c**

Structure d'un programme C

Mon premier programme : Bonjour tout le monde

- La machine ne comprend que le langage machine
- Il faut traduire mon programme programme1.c en langage machine à l'aide d'un traducteur du langage C vers le langage machine
- Un programme appelé compilateur vérifie la syntaxe de mon programme (on dit d'une façon générale, code source) et le traduit en code objet, compris par le processeur
- Le programme en code objet ainsi obtenu peut être exécuté sur la machine

Compilation d'un programme C

Schéma simplifié de la compilation

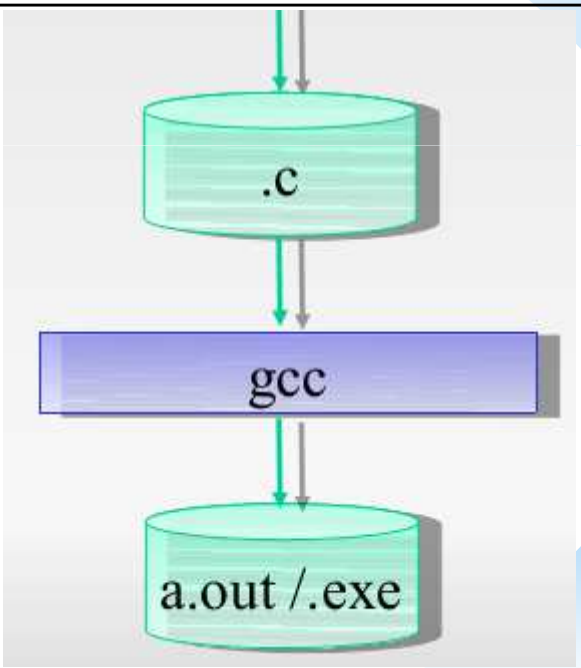
Editeur de texte ou environnement de développement

```
#include <stdio.h>
void main()
{
printf(" Bonjour tout le monde ") ;
}
```

Fichier code source en langage C

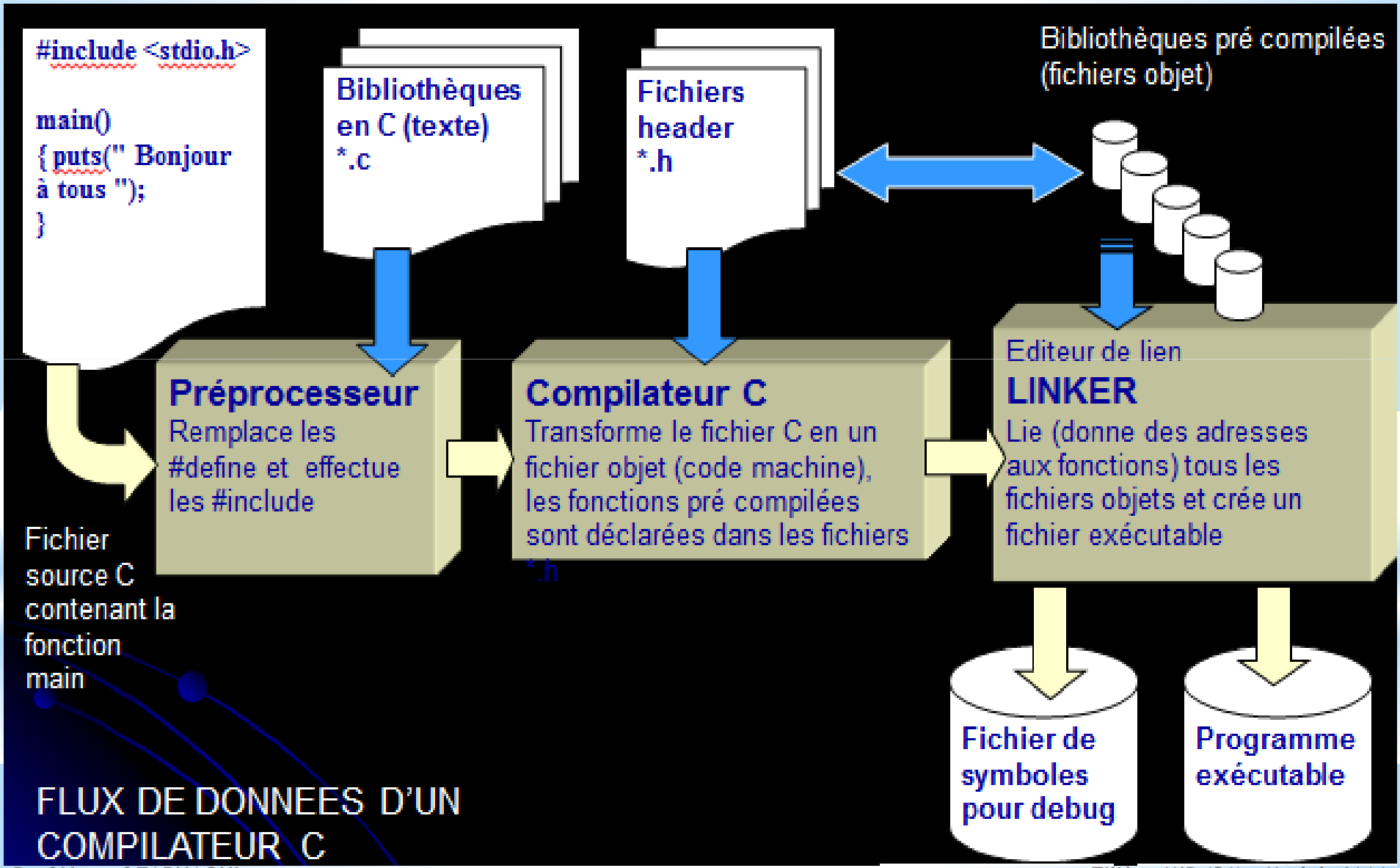
Compilateur

Fichier exécutable



Compilation d'un programme C

Schéma: Bibliothèques de fonctions et compilation



Compilation d'un programme C

Quelques environnements de développement

Sous Windows

- Eclipse
- Netbeans
- Visual C++
- Turbo c++
- Dev-C++

Sous Linux

- Eclipse
- Netbeans
- KDevelop
- …

De l'algorithmique au C

Algorithme et programmation C

Algorithme somme	Entête	#include <stdio.h>
variable X, Y: Entier	Déclarations	void main ( ){
Début		int X, Y ;
X←4	Corps	X=4 ;
Ecrire("Donner Y ")		printf("Donner Y");
Lire(Y)		scanf("%d",&Y);
Ecrire(X+Y)		printf("%d",X+Y);
Fin		}

De l’algorithmique au C

Traduction de l’entête d’un algorithme

Syntaxe en pseudo-code:	Syntaxe en langage C :
Algorithme <nom_algorithme>	void main ( ) (éventuellement ajouter au début du fichier #include<stdio.h>)

De l'algorithmique au C

Traduction des déclarations d'un algorithme : variables

Syntaxe en pseudo-code:

variable<nom_variable> : <type_variable>

Syntaxe en langage C :

<type_variable> <nom_variable> ;

Ne pas oublier le point virgule ;

Exemple :

Syntaxe en algo	Syntaxe en C
variable X : Entier	int X;
variable Y, Z : Réel	float Y, Z;

De l'algorithmique au C

Traduction des déclarations d'un algorithme : variables

Types en algorithmique	Types en langage C
Booléen	Type **non défini** en C mais on peut avoir des expressions booléennes (0 ou différent de zéro)
Entier	**short** = 16 bits **int** = taille du mot machine **long** = 32 bits
Réel	**float** = virgule flottante à simple précision **double** = virgule flottante à double précision
Caractère	**char** = 8 bits
Chaîne de caractères	Type non défini, mais on utilise les pointeurs **char *** ou les tableaux **char [ ]**

De l'algorithmique au C

Traduction des instructions : affectation

Syntaxe de l'affectation en algo	Syntaxe de l'affectation en C
←	=

Exemple:

Syntaxe en algo	Syntaxe de l'affectation en C
X ← 4	X = 4 ;

Remarque : chaque instruction en C se termine par un ;

De l'algorithmique au C

Traduction des instructions : **écriture**

Syntaxe en algo	Syntaxe en C
Ecrire()	printf()

Exemple :

Type de la valeur	Syntaxe en algo	Syntaxe en C
Une chaîne	Ecrire("Bonjour")	printf("Bonjour") ;
X est entier	Ecrire(X)	printf("%d", X) ;
Y est un réel	Ecrire(Y)	printf("%f", Y) ;
Z est un caractère	Ecrire(Z)	printf("%c", Z) ;
Une expression	Ecrire(" La valeur de X =", X , " et de Y =",Y)	printf(" La valeur de X = %d et de Y =%f ", X, Y);

Pour pouvoir utiliser la fonction **printf()**, il faut ajouter au début de votre fichier :

#include<stdio.h>

De l'algorithmique au C

Exemple : Traduire en C l'algorithme suivant

```
Algorithme echange
Variable A, B, C : Entier
Début
        A ← 3
        B ← 2
        Ecrire(" Avant échange")
        Ecrire(" La valeur de A =", A , " et de B =", B)
        C ← B
        B ← A
        A← C
        Ecrire(" Après échange")
        Ecrire(" La valeur de A =", A , " et de B =", B)
Fin
```

De l'algorithmique au C

Corrigé

```
Algorithme echange
Variable A, B, C : Entier
Début
    A ← 3
    B ← 2
    Ecrire(" Avant échange")
    Ecrire(" La valeur de A =", A , " et de B =", B)
    C ← B
    B ← A
    A← C
    Ecrire(" Après échange")
    Ecrire(" La valeur de A =", A , " et de B =", B)
Fin
```

```
#include<stdio.h>
Void main ( )
{
int A, B, C;
A = 3;
B = 2 ;
printf(" Avant échange");
printf("La valeur de A =%d et de B =%d",A,B);
C = B ;
B = A ;
A = C ;
printf(" Après échange") ;
printf("La valeur de A =%d et de B =%d",A, B);
}
```

De l'algorithmique au C

Traduction des instructions : lecture

Syntaxe en algo	Syntaxe en C
Lire()	scanf()

Exemple :

Type de la valeur	Syntaxe en algo	Syntaxe en C
X est entier	Lire(X)	scanf("%d", &X) ;
Y est un réel	Lire(Y)	scanf("%f", &Y) ;
Z est un caractère	Lire(Z)	scanf("%c", &Z) ;

Pour pouvoir utiliser la fonction **scanf()**, il faut ajouter au début de votre fichier :

#include<stdio.h>

De l'algorithmique au C

Exemple : Traduire en C l'algorithme suivant

```
Algorithme echange
Variable A, B, C : Entier
Début
        Ecrire("Donner A")
        Lire(A)
        Ecrire("Donner B")
        Lire(B)
        C ← B
        B ← A
        A← C
        Ecrire("Après échange")
        Ecrire("La valeur de A =", A , " et de B =", B)
Fin
```

De l'algorithmique au C

Corrigé

```
Algorithme echange
Variable A, B, C : Entier
Début
    Ecrire("Donner A")
    Lire(A)
    Ecrire("Donner B")
    Lire(B)
    C ← B
    B ← A
    A← C
    Ecrire("Après échange")
    Ecrire("La valeur de A =", A , " et de B =", B)
Fin
```

```
#include<stdio.h>
Void main ( )
{
int A, B, C;
printf("Donner A");
scanf("%d",&A);
printf("Donner B");
scanf("%d ",&B) ;
C = B ;
B = A ;
A = C ;
printf("Après échange") ;
printf("La valeur de A =%d et de B =%d",A, B);
}
```

Les Constantes caractères (Séquences d'échappement)

Une séquence d'échappement est un couple de symboles dont le premier est le *signe d'échappement* '\'.

\a	**sonnerie**	****	**trait oblique**
\b	curseur arrière	**\?**	point d'interrogation
\t	tabulation	**\'**	apostrophe
\n	nouvelle ligne	**\"**	guillemets
\r	retour au début de ligne	**\f**	saut de page (imprimante)
\0	NUL	**\v**	tabulateur vertical

Les opérateurs standards

Opérateurs arithmétiques

+	addition
-	soustraction
*	multiplication
/	division (entière et rationnelle!)
%	modulo (reste d'une div. entière)

Opérateurs logiques

&&	**et logique**
||	ou logique
!	négation logique

Opérateurs de comparaison

==	**égal à**
!=	différent de
<, <=, >, >=	plus petit que, ...

Initialisation des variables

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

Exemple:

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

Remarque:

int A=10; 

```
int   A;
A=10;
```

Initialisation des variables

Les constantes

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

Exemple:

```
const int    MAX = 767;
const double TVA = 0.25;
const double   e = 2.7;
const char   NEWLINE = '\n';
```

Commentaires

Les commentaires sont non seulement utiles, mais nécessaires à la compréhension d'un programme.

Forme « standard » : /*... */ ou //…

Exemple:

```
a = a + 1; /* Ceci est un commentaire de ligne */
b = b - 1; /* Et ceci en est un autre */
```

Les opérateurs particuliers de C

En pratique, nous retrouvons souvent des affectations comme: **i = i + 2**

En C, nous utiliserons plutôt la formulation plus compacte: **i += 2**

L'opérateur **+=** est un *opérateur d'affectation*.

Pour la plupart des expressions de la forme:

expr1 = (expr1) op (expr2)

Il existe une formulation équivalente qui utilise un opérateur d'affectation:

expr1 op= expr2

Les opérateurs particuliers de C

Opérateurs d'affectation

+=	ajouter à
-=	diminuer de
*=	multiplier par
/=	diviser par
%=	modulo

Exemple

A+=2; ⟺ A=A+2;

A*=B ⟺ A=A*B;

A%=B ⟺ A=A%B;

Opérateurs d'incrémentation et de décrémentation

Les affectations les plus fréquentes sont du type: **I = I + 1** **et** **I = I - 1**
En C, nous disposons de deux opérateurs pour ces affectations:

I++ ou ++I	**pour l'incrémentation (augmentation d'une unité)**
I-- ou --I	pour la décrémentation (diminution d'une unité)
X=I++	passe d'abord la valeur de I à X et incrémente après
X=I--	passe d'abord la valeur de I à X et décrémente après
X=++I	incrémente d'abord et passe la valeur incrémentée à X
X=--I	décrémente d'abord et passe la valeur décrémentée à X

var ++; ⇒ Post-incrémentation
++ var; ⇒ Pré-incrémentation

Exemple
Supposons que la valeur de N est égal à 5:

X = N++;	**Résultat: N=6 et X=5**
X = ++N;	Résultat: N=6 et X=6

Les priorités des opérateurs

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

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>

COMMANDE C	EXPLICATION
exp(X)	fonction exponentielle
log(X)	logarithme naturel
log10(X)	logarithme à base 10
pow(X,Y)	X exposant Y
sqrt(X)	racine carrée de X
fabs(X)	valeur absolue de X
sin(X) cos(X) tan(X)	**sinus, cosinus, tangente de X**
asin(X) acos(X) atan(X)	arcsin(X), arccos(X), arctan(X)
sinh(X) cosh(X) tanh(X)	sinus, cosinus, tangente hyperboliques de X

Les conversions de type

Les conversions de type automatiques

les valeurs des opérandes sont ***converties automatiquement dans un type commun***. Ces manipulations implicites convertissent en général des types plus 'petits' en des types plus 'larges';

char < short ≤ int ≤ long < float < double

Exemple

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

- Pour le calcul de *D*, *A* est converti en **float** et divisé par *C*. Le résultat (0.75) est affecté à *D* qui est aussi du type **float**. On obtient donc: *D=0.75*
- Pour le calcul de *E*, *A* est converti en **int** et divisé par *B*. Le résultat de la division (type **int,** valeur 0) est converti en **float**. On obtient donc: *E=0.000*
- Pour le calcul de *F*, *A* est converti en **float** et divisé par *C*. Le résultat (0.75) est retraduit en **char**. On obtient donc: *F=0*

Les conversions de type

Les conversions de type forcées (casting)

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

(<Type>) <Expression>

Exemple

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

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

Résultat: *C=0.75*

Écriture d'un caractère *putchar('a');*

putchar() c'est une fonction d'écriture d'un caractère.

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               */
```

Remarque:

putchar(B); ⟺ **printf("%c",B) ;**

Lecture d'un caractère *getchar*

getchar() c'est une fonction de lecture d'un caractère

Exemple:

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

```
Scanf("%c",&c);
```

Exemple 1 : Traduire en C l'algorithme suivant

Algorithme Calcul
Variable A : Entier
C,B : Réel
D : caractère
E : Booléen
Début
A ← 30
B ← A * 2
Écrire('B=' , B)
C ← (B + A)/4
B ← C / 5
D ← ‘A’
E ← (A > 40) Ou (C < B)
Écrire('les valeurs obtenues sont : A = ' , A , 'B = ' ,B , ' C =’,C, ' D = ', D, ' E = ', E)
Fin

Exemple 2 : Traduire en C l'algorithme suivant

Algorithme Surface d'un cercle

Constante Pi←3.14

Variable Rayon **: Entier** * Donnée d'entrée*

Variable Surface **: Réel** * Donnée de sortie*

DEBUT

Écrire ('Saisir la valeur du rayon' **)**

Lire(Rayon)

Surface ⟵ Rayon * Rayon * Pi

Écrire (' La Surface du cercle est : **',** Surface**)**

FIN

Exemple 3 : Traduire en C l'algorithme suivant

Algorithme Commission

Constante M ← 4000 * M: ***montant fixe****

Variable CA : Entier * Donnée d'entrée (CA: ***chiffre d'affaire)*** *

Com : Réel * Donnée intermédiaire (Com: ***commission)****

Sal : Réel * Donnée de sortie(Sal: ***salaire mensuel***)*

DEBUT

Écrire ('Donner le CA mensuel en DHS'**)**

Lire(CA)

Com ←CA * 10/100

Sal ← Com + M

Écrire ('Le salaire mensuel est de : ', Sal, ' en DHS ' **)**

FIN