Langage C



Le langage C
Sommaire :

Chapitre 1: Notion de base. 5
1. Introduction : 5
2. Caractéristiques du langage C: 6
3. Environnement du compilateur en C: 6
4. Bases de la programmation en C : 8
4.1 Structure d'un programme 8
4.2 Fonctions pré-compilées et headers files 9
4.3 La phase de compilation. 9
4.4 Les commentaires en C : 10
Chapitre 2: Variables, Constantes, Opérateurs et les fonctions d'affichages et de saisie. 12
1. Variables et Constantes : 12
1.1. Définitions : 12
1.2. Représentation mémoire : 12
1.3. Types de Variables ou de Constantes : 13
1.3.1 Les Entiers : 13
1.3.2 Les Caractères : 13
1.3.3 Les Chaînes de Caractères : 15
1.3.4 Les Booléens : 15
1.3.5 Les Réels : 15
1.4. Les Constantes : 15
1.5. Les Variables : 16
1.5.1 La classe : 16
1.5.2 L'identificateur: 16
1.5.3 Le type: 16
1.5.4 Déclaration de variables : 17
1.6 Déclaration des Variables et des Constantes 17
2. Les Opérateurs : 18
2.1 L'affectation : 18
2.2 Les opérateurs Arithmétiques : 19
2.3 Les opérateurs d'Incrémentation et de Décrémentation : 19
2.4 Les opérateurs Binaires : 19
2.5 Les opérateurs composés : 20
2.6 Les opérateurs relationnels : 20
2.7 Les opérateurs logiques : 20
2.8 Priorité des opérateurs : 21

3 Les fonctions d'affichages et de saisie 21
3.1 Fonction d'affichage : printf 21
3.1.1 Format d'affichage 22
3.1.2 Code de contrôle 22
3.2 Fonction de Saisie : scanf 24
4 L'opérateur de conversion de type. 26
5 Quelques exercices sur le chapitre 1 & 2 : 27
Chapitre 3: Structures de contrôle. Verif exo 32
1 Branchement conditionnel : 32
1.1 Si … Alors… : 32
1.2 Si … Alors… Sinon : 32
1.3 Si … Alors… Sinon Si : 33
2 Boucles : 34
2.1 Tant que… Faire…: 34
2.2 Faire… Tant que…: 35
2.3 Faire… jusqu'à…: 35
3 Instructions de contrôle : 36
3.1. Instruction de sortie : break 36
3.2. Instruction continue : 37
3.3. Instruction de branchement goto : 37
4 Structure de choix : switch 38
5 Quelques Exercices : 39
Chapitre 4: Les Fonctions. 40
1 Introduction: 40
2 Déclaration : 41
2.1 La mise en place d'une fonction dans un programme. 41
2.2 Prototypage : 42
2.3 Définition d'une fonction sans valeur retournée (Procédure) : 43
2.4 Appel d'une fonction sans valeur retournée (procédure) : 43
2.5 Définition d'une fonction avec valeur retournée : 46
2.6 Appel d'une fonction : 46
3 Quelques Exercices : 49
Chapitre 5: Les Tableaux. 50
1 Introduction: 50
2 Utilisation des tableaux dans le cadre scientifique : 51
2.1. Notion d'adresse : 51
2.2. Initialisation de tableaux : 52
2.3. Remplissage de tableaux : 53
2.4. Passage en paramètres à une fonction : 53
2.5. Tableau à plusieurs dimensions : 54
3 Les chaînes de caractères : 55

3.1 Déclaration : 55
3.2 Initialiser des chaînes de caractère : 55
3.3 L'affichage d'une chaînes de caractère se fait aussi de plusieurs manières : 56
4 Quelques Exercices : 57
Chapitre 6: Les Structures. 58
1 Introduction 58
2 Utilisation des structures : 58
2.1 La mise en place d'une structure dans un programme. 58
2.2 Déclaration d'une structure : 59
2.3 Accès aux champs d'une structure : 59
2.4 Tableau de structure : 61
3 Les déclarations de type synonyme: *typedef* 62
4 Quelques Exercices : 64
Chapitre 7: Les Pointeurs. 65
1 Introduction 65
2 Déclaration de pointeur: 65
5 Arithmétique sur les pointeurs : 66
6 Affectation d'une valeur à un pointeur : 68
7 Tableaux de pointeur : 69
8 Allocation dynamique de mémoire 71
9 Les pointeurs et les structures de données 73
10 Quelques Exercices : 75
Chapitre 8: Les Fichiers. 76
1 Introduction 76
2 Déclaration: 77
3 Ouverture et fermeture d'un fichier 77
3.1 Ouverture d'un fichier: fopen 77
3.2 Fermeture d'un fichier: fclose 80
4 Lecture et écriture sur un fichier : 81
4.1 Lecture et écriture par caractères : 81
4.2 Lecture et écriture par chaîne de caractères : 83
4.3 Lecture et écriture avec format : 85
5 Opération en mode binaire : 87
5.1 Lecture et écriture : 87
5.2 Positionnement dans un fichier : 89
6 Exercices 92

Fin du sommaire

Chapitre 1: Notion de base.

1. Introduction :

Le Langage C a été créé à la fin des années 70 aux USA. Il a été conçu à l'origine pour être le langage de programmation du système d'exploitation UNIX. Ses créateurs KERNIGHAN et RITCHIE l'ont défini dans un livre intitulé « The C Programming Language » ou plus communément « K&R 78 ».

A la fin des années 80 l' American National Standard Institute (ANSI) développa une version standardisée du C. En 1990 l' « International Standards Organization »(ISO) adopta la standard ANSI sous le nom ISO C, appelé plus communément l'ANSI C.

Toutes les implémentations actuelles du C obéissent au standard de l'ANSI-C

C est un langage très largement diffusé, il est l'outil de base des Electroniciens.

Un programme n'est autre qu'une suite d'instructions intelligibles par l'ordinateur traitant les données acquises du monde extérieur, les modifiant, les stockant ou les imprimant.

L'ordinateur ne comprend qu'un langage qui lui est spécifique appelé langage machine (suite de 0 et de 1) mais puisque la programmation en langage machine est longue et compliquée des langages évolués ont été créé pour faciliter le travail du programmeur. Le programme écrit en langage évolué sera traduit en langage machine par un programme particulier appelé compilateur.

Programme évolué ⟹ compilateur ⟹ programme machine

2. Caractéristiques du langage C:

Comme déjà évoqué la popularité du C est due principalement à ses caractéristiques:

✓ ***Universalité:*** Le langage C n'est pas spécifique à un domaine d'applications précis tel que COBOL (pour les applications mettant en jeu de grandes quantités de données) ou FORTRAN (applications scientifiques), par contre son utilisation est possible pour tout type d'applications.

✓ ***Simplicité*** **:** Le noyau du langage C est formé d'un nombre limité mais efficace d'opérateurs et de fonctions.

✓ ***Modernité*** **:** La programmation en C peut être qualifiée de moderne du fait que C est un langage structuré.

✓ ***Portabilité:*** Il existe des compilateurs C pour tout type de machines et de systèmes d'exploitation, ainsi un programme C peut être utilisé sur différents ordinateurs simplement en le recompilant (notez qu'avec les environnements graphiques de plus en plus communs la portabilité devient presque impossible).

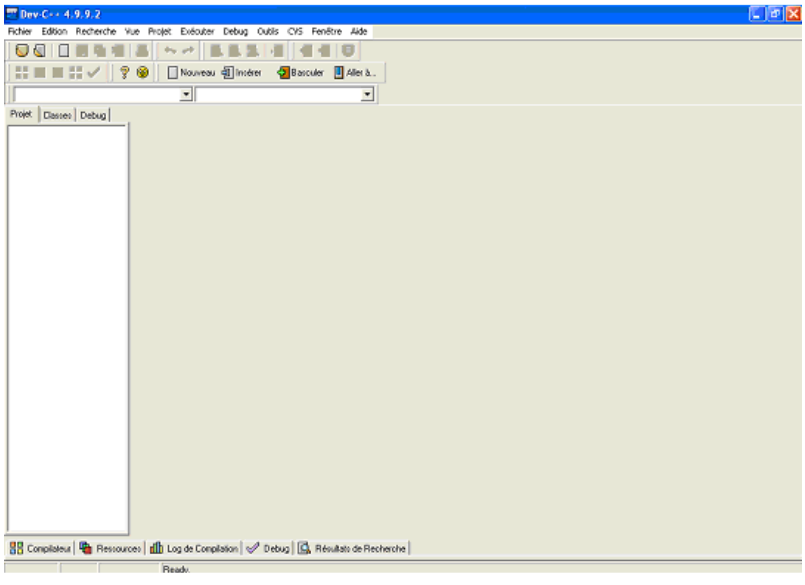
3. Environnement du compilateur en C:

L'environnement du compilateur C++ utilisé pour le langage C requière quelques notions pour éviter des erreurs dans les différentes applications futures.

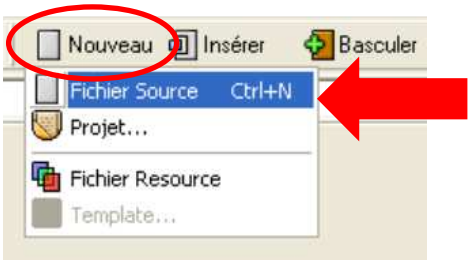
Cliquer sur l'icône suivant :



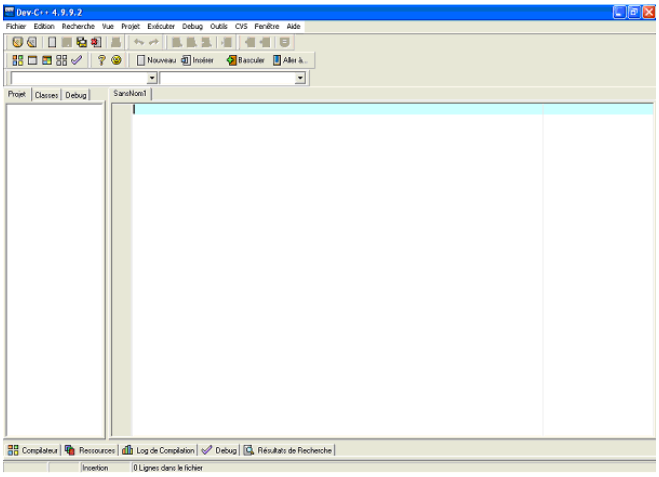
Et on a la fenêtre suivante :



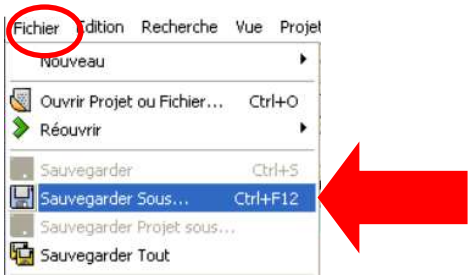
Allez dans nouveau fichier source :



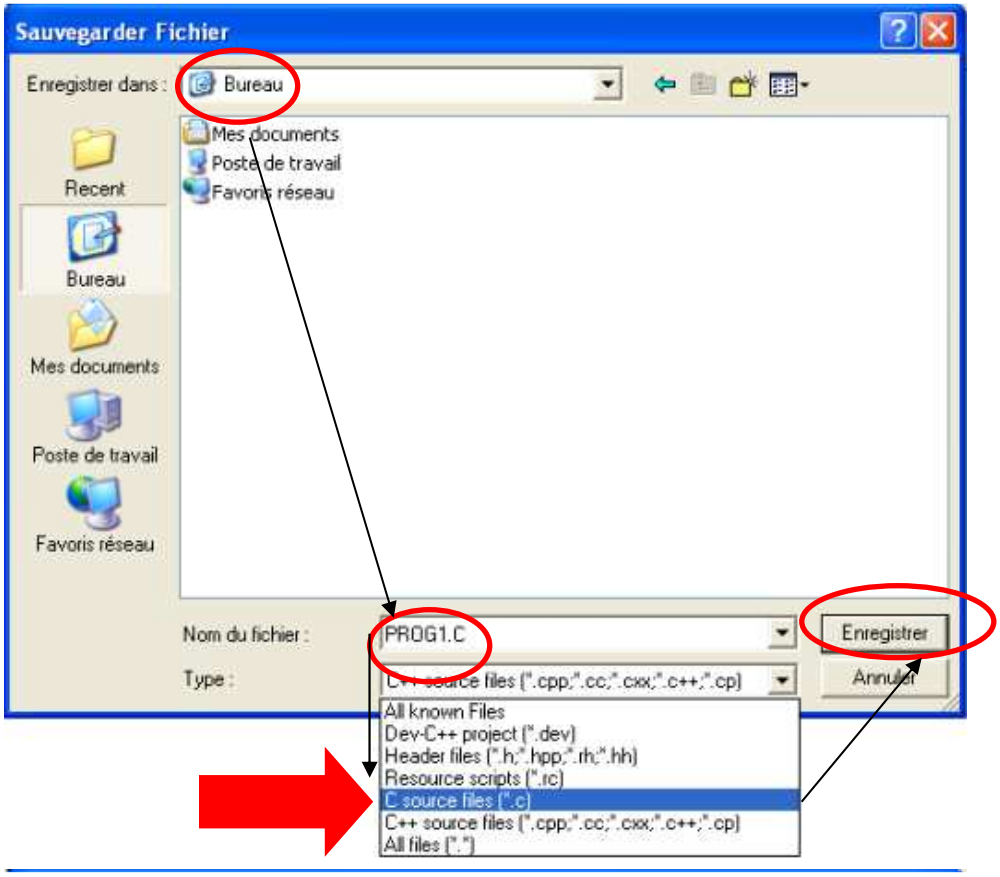
Et là nous avons notre page pour écrire les différentes ligne de code en C :



Donnez un nom a votre futur programme :



Et une nouvelle fenêtre apparaît :



Remarques :

-Le nom du fichier ne doit pas comporter des caractères spéciaux et il ne doit pas dépasser 8 caractères.
-Le fichier doit être en **«*.C »**.

4. Bases de la programmation en C :

4.1 Structure d'un programme

Un programme écrit en C doit obéir à certaines règles bien définies. Le langage reconnaît uniquement la structure de fonction: un programme écrit en C est un ensemble de fonctions dont chacune peut s'appeler elle-même et appeler les autres fonctions. Une fonction particulière, dont la présence dans le programme est obligatoire, est la fonction **main**. Le programme commencera toujours son exécution au début de cette fonction. La structure la plus simple d'un programme C est alors :

Exemple :

(1) Déclaration d'une ou plusieurs librairie(s)	#include <stdio.h> #include <math.h>
(2) Déclaration de constantes et des variables globales et les commandes pour le pré-processeur	int i; // une variable globale de type entier

(3) Programme principal	main()
	{
	
	// Le programme
	
	}

Remarquez les parenthèses obligatoires après le nom de la fonction (ici main) et les accolades d'ouverture et de fermeture délimitant le début et la fin de la fonction.

4.2 Fonctions pré-compilées et headers files

Il existe en C des fonctions pré compilées qui peuvent être incluses dans nos programmes, ces fonctions sont contenues dans des fichiers spéciaux appelés library files (extension *.lib). Pour accéder à ces fonctions une directive doit être issue au compilateur indiquant d'inclure les header files (extension *.h) contenant les déclarations correspondantes à ces fonctions. La directive doit être écrite en début de programme. Sa syntaxe est la suivante:

Syntaxe: **#include <fichier.h>**

Les caractères < et > indiquent que le fichier se trouve dans le répertoire par défaut des fichiers d'en tête.

Syntaxe: **#include "fichier.h"**

Le fichier se trouve dans le répertoire courant utilisateur.

On peut aussi inclure un fichier personnel écrit en langage C, dans ce cas il faudra indiquer le chemin absolu.

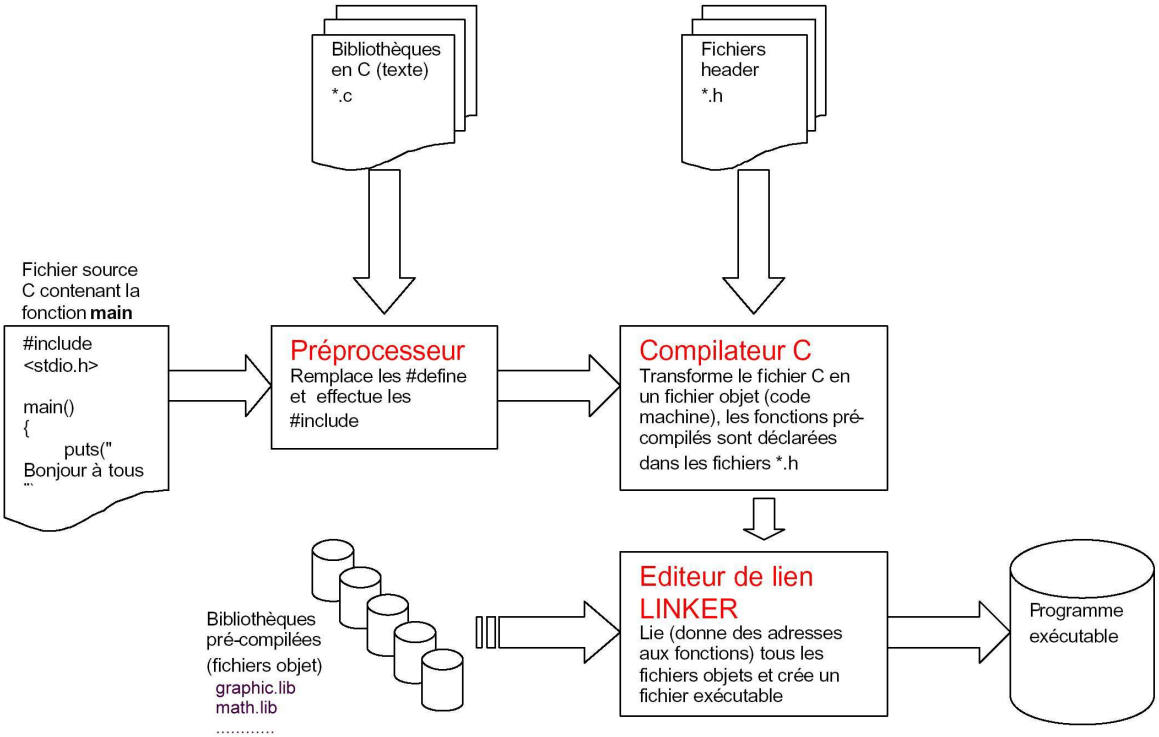
Syntaxe: **#include "C:\REP\TOTO.H"**

Attention: Il n'y a pas de ; à la fin de cette directive et d'une manière générale à la fin de toutes les instructions qui commencent par # (directives réservées au préprocesseur).

4.3 La phase de compilation.

Les compilateurs C réalisent deux transformations du programme :

- Un pré-processeur réalise des transformations d'ordre purement textuel, pour rendre des services de type inclusion de sources.
- Le compilateur récupère le texte généré par le pré-processeur et le traduit en langage machine (instructions).



4.4 Les commentaires en C :

Dans un programme C, les commentaires sont nécessaires. Ils permettent de faciliter la lecture et la compréhension du programme par le programmateur et les autres personnes.
Le début d'un commentaire est marqué par **/*** et la fin par ***/**. Notons que les commentaires imbriqués sont interdits. Par ailleurs, le compilateur ne tiendra pas compte de tout texte inscrit entre cette syntaxe.

Exemples : /* Voici
Un
Commentaire */
Le commentaire est inscrit sur plusieurs lignes. Le compilateur attends le signe */ pour clôturer le commentaire.

Avec la syntaxe // le commentaire est contenu dans la ligne. Le compilateur n'attends pas de caractère de fin de commentaire. La ligne suivante sera une nouvelle ligne de programme.
Exemple : // Un autre commentaire

Application :

```
#include<stdio.h>
main( )
{
//commentaire 1

/* commentaire
numéro
2
*/
system("pause") ;
}
```

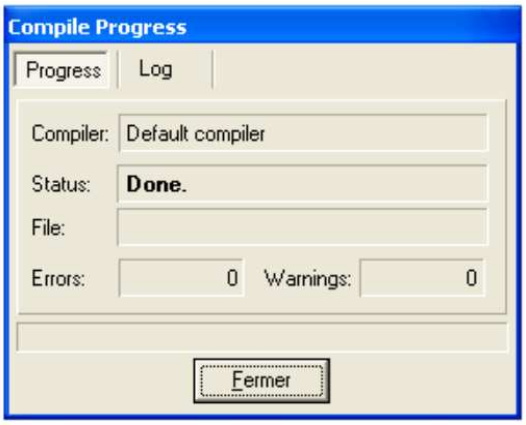
Commentaire(s)

Une fois le programme tapé utiliser les icônes suivants :

1ére méthode :

Cliquez sur : 
Cela permet de compiler votre programme et de donner le nombre d'erreur ou de warning dans votre programme.

Le résultat est mis sous la forme suivante :



Maintenant vous pouvez cliquer sur fermer et ensuite sur icône suivant :



Cela vous exécute le programme dans une page DOS

2ème méthode :

Cliquez sur : 

Cette action permet en une seule fois de compiler et d'exécuter le programme.

Il est recommandé d'utiliser cette seconde méthode pour gagner du temps lorsque l'on fait des petites modifications dans un programme.

Bien entendu ce type de manipulation sera à faire pour chaque application.

Deuxième application :

```
#include<stdio.h>
main( )
{
//commentaire 1

/* commentaire
numéro
2
*/
}
```

Commentaire(s) :
Que remarquez-vous ?

Chapitre 2: Variables, Constantes, Opérateurs et les fonctions d'affichages et de saisie.

1. Variables et Constantes :

1.1. Définitions :

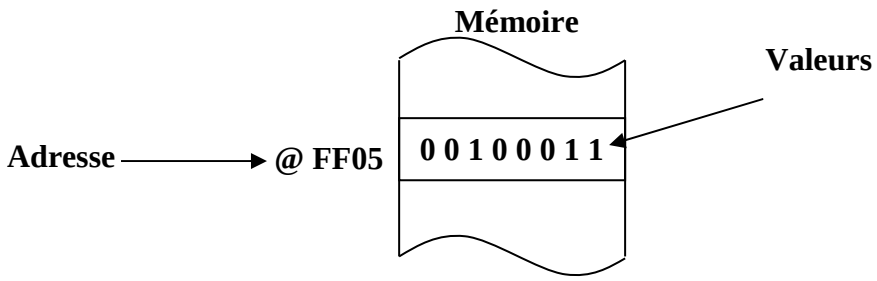
Constante : Une constante ne change jamais de valeurs durant l'exécution du programme. Elle est généralement stockée dans la mémoire morte de la machine (ROM).

Variable : Une variable peut changer de valeur durant l'exécution du programme. Elle est généralement stockée dans la mémoire vive de la machine.

Une variable ou une constante est souvent définie par cinq éléments :

- L'identificateur: C'est le nom que l'on donne à la variable ou à la constante.
- Le type: Si la variable est un entier ou un caractère ou une chaîne de caractère ou un réel ou un booléen.
- La taille: C'est le nombre d'octets occupés en mémoire, elle est fonction du type.
- La valeur: C'est la valeur que l'on attribut à la variable ou à la constante.
- L'adresse: C'est là où sont stocké la valeur de la variable ou de la constante. L'adresse peut également être appelée Pointeur.

1.2. Représentation mémoire :



1.3. Types de Variables ou de Constantes :

Les Types permettent de définir si la lettre A dans un programme et définie comme un nombre, une lettre etc.

1.3.1 Les Entiers :

Un entier est un nombre positif ou négatif. *Exemple : + 3025, - 852*
Le mot clé désignant un entier est ***int***. Il peut être affecté de deux types d'attributs : Un attribut de précision (***short, long***) et un de représentation (***unsigned***).
Les constantes entières peuvent s'écrire en Décimal (***a=10***), Hexadécimal (***a=0x0A***), Octal (***a=012***), Binaire (***a=1010***).

Base décimale	Base octale	Base hexa	Représentation binaire
15			
	01000		
		0xff0	
			010111000101
200			

1.3.2 Les Caractères :

Un caractère est un nombre entier positif compris entre 0 et 255 ; c'est le code ASCII d'un caractère.
Le mot-clé désignant un caractère est ***char***.

Attention : Ne pas confondre le code ASCII et l'octet.
Par exemple le chiffre 1 a pour valeur en hexa (char Var=0x31 ; ou Var='1' ;) et le chiffre 1 a pour valeur en octet (char Var=1 ;//Valeur en hexa 0x01).

Tableau du code ASCII.

Code	Caractère	Code	Caractère	Code	Caractère	Code	Caractère	Code	Caractère
0	[car. nul]	69	E	116	t	164	¤	211	Ó
…		70	F	117	u	165	¥	212	Ô
7	[sig. sonore]	71	G	118	v	166	¦	213	Õ
8	[ret. arrière]	72	H	119	w	167	§	214	Ö
9	[tabulation]	73	I	120	x	168	¨	215	×
10	[saut ligne]	74	J	121	y	169	©	216	Ø
11	[tab. vert.]	75	K	122	z	170	ª	217	Ù
12	[saut page]	76	L	123	{	171	«	218	Ú
13	[ret. chariot]	77	M	124	|	172	¬	219	Û
...		78	N	125	}	173	-	220	Ü
32	[espace]	79	O	126	~	174	®	221	Ý
33	!	80	P	…		175	¯	222	Þ
34	"	81	Q	128	€	176	°	223	ß
35	#	82	R	…		177	±	224	à
36	$	83	S	130	‚	178	²	225	á
37	%	84	T	131	*f*	179	³	226	â
38	&	85	U	132	„	180	´	227	ã
39	’	86	V	133	…	181	µ	228	ä
40	(	87	W	134	†	182	¶	229	å
41	)	88	X	135	‡	183	·	230	æ
42	*	89	Y	136	ˆ	184	¸	231	ç
43	+	90	Z	137	‰	185	¹	232	è
44	,	91	[	138	Š	186	º	233	é
45	-	92	\	139	‹	187	»	234	ê
46	.	93	]	140	Œ	188	¼	235	ë
47	/	94	^	…		189	½	236	ì
48	0	95	_	142	Ž	190	¾	237	í
49	1	96	`	…		191	¿	238	î
50	2	97	a	145	‘	192	À	239	ï
51	3	98	b	146	’	193	Á	240	ð
52	4	99	c	147	“	194	Â	241	ñ
53	5	100	d	148	”	195	Ã	242	ò
54	6	101	e	149	•	196	Ä	243	ó
55	7	102	f	150	–	197	Å	244	ô
56	8	103	g	151	—	198	Æ	245	õ
57	9	104	h	152	˜	199	Ç	246	ö
58	:	105	i	153	™	200	È	247	÷
59	;	106	j	154	š	201	É	248	ø
60	<	107	k	155	›	202	Ê	249	ù
61	=	108	l	156	œ	203	Ë	250	ú
62	>	109	m	…		204	Ì	251	û
63	?	110	n	158	ž	205	Í	252	ü
64	@	111	o	159	Ÿ	206	Î	253	ý
65	A	112	p	160	[espace]	207	Ï	254	þ
66	B	113	q	161	¡	208	Ð	255	ÿ
67	C	114	r	162	¢	209	Ñ		
68	D	115	s	163	£	210	Ò		

1.3.3 Les Chaînes de Caractères :

Une chaîne de caractères est une suite de caractères mis les uns à la suite des autres afin de former un mot.
Le mot clé désignant la chaîne est également ***char***.

Exemple : "Police" est la suite des caractères 'P','o','l','i','c' ,'e' ; ce qui correspond au code ASCII 80, 111, 108, 105, 99, 101.
char var= " Police" ; *//Déclaration de la variable var à la valeur Police*
Rem : Le compilateur ajoute en fin de chaîne un caractère 0 (*null en code ASCII*). Ainsi le compilateur rangera en mémoire la chaîne « Police » sous la forme de 7 caractères P, o, l, i, c, e, \0.

1.3.4 Les Booléens :

Le booléen ne peut prendre que deux valeurs VRAI ou FAUX.

1.3.5 Les Réels :

Un réel est un nombre à virgule positif ou négatif avec un exposant.
Exemple : $12,344.10^{e}-5$
float var = 12.344e-5; //Déclaration de la variable var à la valeur 0,00012344

1.4. Les Constantes :

Les constantes n'existent pas, c'est à dire qu'il n'y a pas d'allocation mémoire, mais on peut affecter à un identificateur (Nom) une valeur constante par l'instruction #define. Le pré processeur se chargera de remplacer les constantes par leurs valeurs.

Syntaxe : **<#define> <identificateur> <valeur>**

Exemple:
#define PI 3.14

1.5. Les Variables :

Les variables sont définies par *la classe, le type et l'identificateur.*

Syntaxe : **<classe> <type> <identificateur 1>, ……, <identificateur n> ;**

1.5.1 La classe :

La classe détermine le comportement des variables pendant l'exécution d'un programme, très peu utilisée. Elle est très souvent supprimée dans la déclaration de variable.

Classe	*Signification*
Auto	Classe par défaut.
Register	Force le compilateur à utiliser un registre interne du microprocesseur dans la mesure du possible.
Static	La valeur est conservée dans tout le programme.
Externe	La variable es accessible par d'autres programme.

1.5.2 L'identificateur:

C'est le *nom* affecté à la variable. Le nombre de caractères peut être limité, cela dépend du compilateur utilisé.

1.5.3 Le type:

Il détermine la taille de la variable et les opérations pouvant être effectuées. Le type est fonction de la machine ou du compilateur utilisé. On peut rajouter le mot unsigned devant le type du variable, alors la variable devient non signée et cela permet d'étendre la plage de valeurs.

1.5.4 Déclaration de variables :

Type	*Taille (En Bits)*	*Signé*	*Non Signé (Unsigned)*
Caractère->**Char**	8	-128 à +127	0 à +255
Entier Court->**Short**	16	-32768 à +32768	0 à +65535
Entier->**Int**	16 (ou 32)	-32768 à +32768	0 à +65535
Entier Long->**Long**	32	-21477483647 à +21477483647	0 à +4294967295
Réel->**Float**	32	$+/-3,4.10^{-38}$ à $+/-3,4.10^{-38}$	Aucune signification
Réel Double précision->**Double**	64	$+/-1,7.10^{-308}$ à $+/-1,7.10^{-308}$	Aucune signification

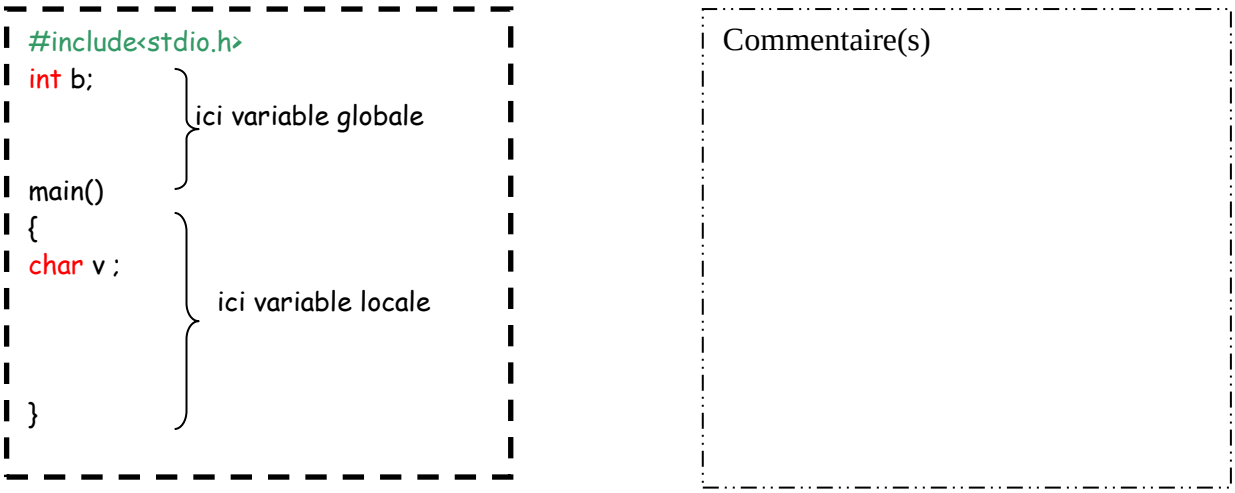
1.6 Déclaration des Variables et des Constantes

Un programme C est une suite de déclarations de variables. Certaines sont accessibles durant toute la vie du programme ; ce sont les *variables globales*. D'autres sont déclarées à l'intérieur d'une fonction et leur durée de vie est limité à l'exécution de cette fonction ; ce sont les *variables locales*.

Variables	Accès	Visibilité	Exemple
GLOBALE	Adresse fixe	Déclarée en dehors d'une fonction, visible partout	char c ;
LOCALE	Pile (perdue à la sortie)	Déclarée et visible dans une fonction	Void fonction(void) { char c ; …
STATIQUE	Adresse fixe	Déclarée et visible dans une fonction	Void fonction(void) { **static** char c ; …
EXTERNE		Déclarée en dehors d'une fonction et initialisée dans une bibliothèque externe	**extern** char c ;

Nous prendrons pour habitude de déclarer les variables globales en début de programme.

Exemple :



2. Les Opérateurs :

Ce sont eux qui régissent toutes les opérations ou transformations sur les valeurs de variables. Un problème courant est de savoir quel est l'opérateur qui est le plus prioritaire sur l'autre. Pour résoudre ce problème il est conseillé de mettre des parenthèses pour bien séparer les différentes opérations.

2.1 L'affectation :

L'affectation est l'opérateur permettant de modifier la valeur d'une variable. Elle est toujours effectuée de la droite vers la gauche. En effet le programme commence par évaluer la valeur de l'expression puis l'affecte avec le signe = à la variable par son identificateur.

=	*<identificateur>=<expression> ;*

Exemple : var= 35; // La variable var prends la valeur 35
i=i+j ; // La variable i prends la valeur de i+j

2.2 Les opérateurs Arithmétiques :

+	*Addition*
-	*Soustraction*
*	*Multiplication*
/	*Division*
%	*Modulo (Reste de Division)*

Rem : Division et multiplication reste prioritaire sur l'addition et la soustraction.

Exemple : var=(5%2)+3; // La variable var prends la valeur 4 ;

2.3 Les opérateurs d'Incrémentation et de Décrémentation :

L'opérateur incrémentation et décrémentation partagent les caractéristiques suivantes :

- Il peut s'utiliser en préfixe ou en postfixe .
- Il s'applique à une variable *var*

Cet opérateur se note -- *var*, décrémente la variable *var* de 1 et délivre la nouvelle valeur.
Cet opérateur se note *var* --, décrémente la variable var de 1 et délivre la valeur initiale.

++	*Incrémente de 1*
--	*Décrémente de 1*

Remarque : Cet opérateur est très utilisé lors de la manipulation de tableau.

2.4 Les opérateurs Binaires :

Ils permettent d'agir sur les bits constituants les variables de type entier.

&	*ET*
|	*OU*
^	*OU (Exclusif)*
~	*NON (Complément à 1)*
>>	*Décalage à droite*
<<	*Décalage à gauche*

2.5 Les opérateurs composés :

Chacun des opérateurs + - * / % >> << & ^ | peut être associer à l'opérateur = afin de former les opérateurs d'affectation composée suivant :

+=	*Addition et affectation*
-=	*Soustraction et affectation*
*=	*Multiplication et affectation*
/=	*Division et affectation*
%=	*Modulo et affectation*
&=	*ET et affectation*
|=	*OU et affectation*
^=	*OU Exclusif et affectation*
<<=	*Décalage à gauche et affectation*
>>=	*Décalage à droite et affectation*

2.6 Les opérateurs relationnels :

Les opérateurs relationnels permettent d'évaluer puis de comparer deux expressions. La valeur rendue est de type int et prends deux valeurs :

- La valeur 1 si la condition est vraie.
- La valeur 0 si la condition est fausse.

On parle ici de valeur booléenne.
Les opérateurs sont les suivants :

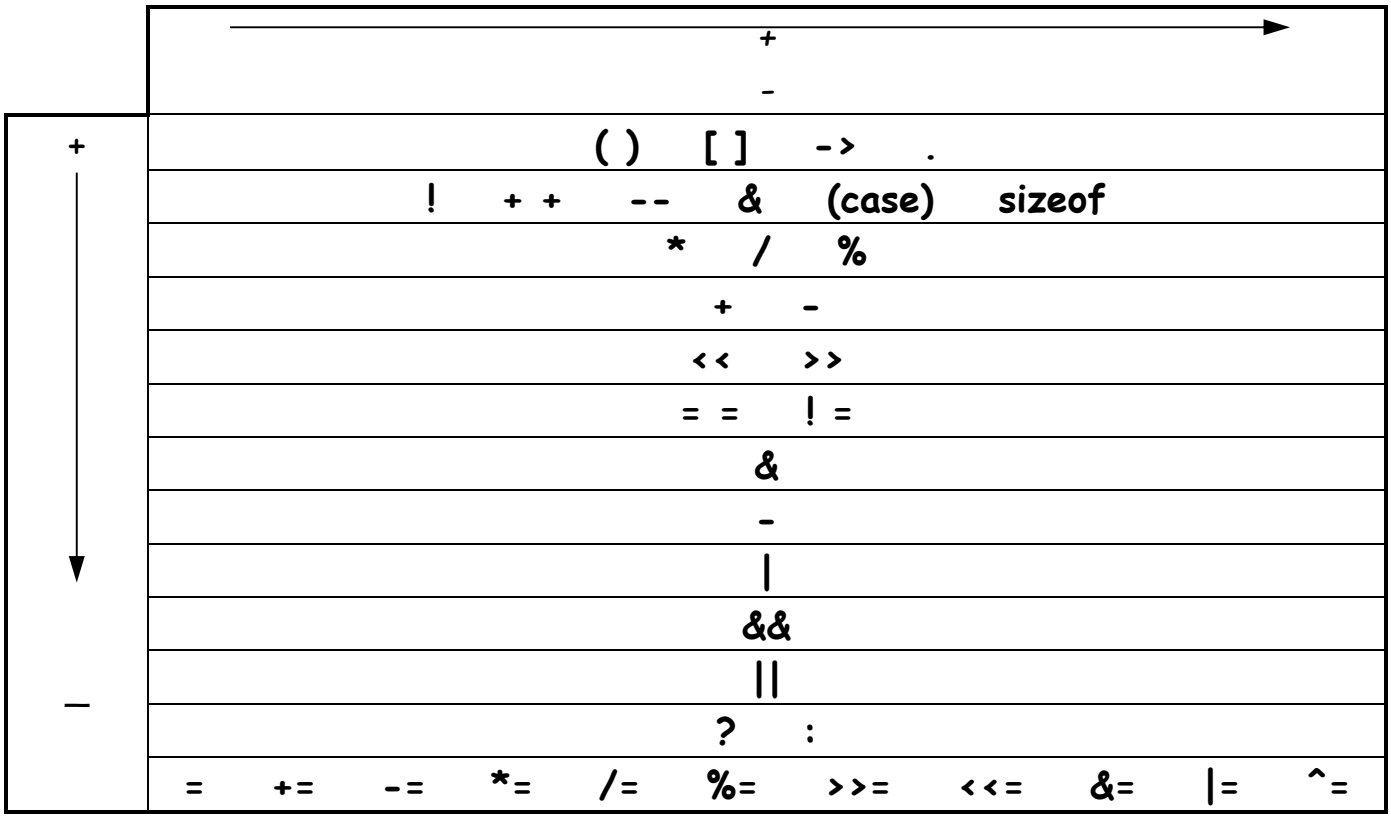
>	*Supérieur à*
>=	*Supérieur ou égal à*
<	*Inférieur à*
<=	*Inférieur ou égal à*
= =	*Egal à*
!=	*Différent*

2.7 Les opérateurs logiques :

Ils sont utilisés comme les opérateurs relationnels :

!	*Négation*
&&	*ET Logique*
| |	*OU Logique*

2.8 Priorité des opérateurs :



3 Les fonctions d'affichages et de saisie

3.1 Fonction d'affichage : printf

Elle permet d'afficher sur l'écran un message.

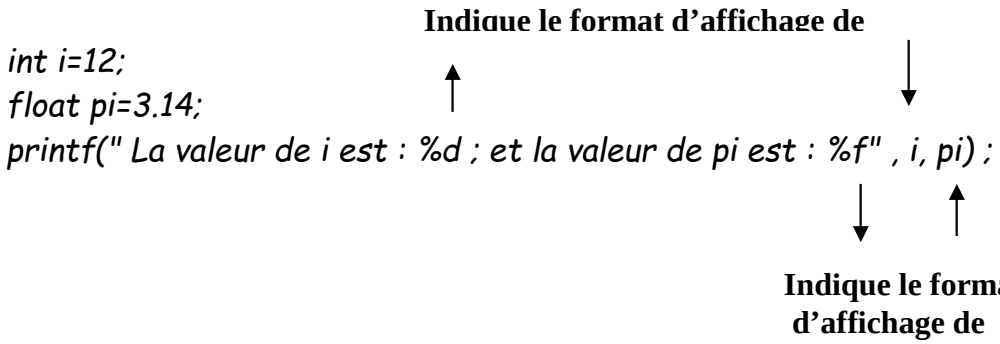
Syntaxe : **Printf (¨chaîne de caractères ¨) ;**

La déclaration de la fonction printf en tant que fonction externe se trouve dans le fichier stdio.h qu'il faudra inclure avant toute utilisation en tout début de programme.

Elle permet aussi de réaliser des sorties formatées de messages et (ou) de valeurs des variables sous différents formats.

Syntaxe : **Printf ("chaîne de caractères ", identificateur1,……,identificateurn) ;**

« chaîne de caractères » est le texte à afficher dans lequel on peut afficher la valeur des expressions se trouvant dans identificateur. Pour cela on va mettre une séquence d'échappement indiquant le format dans lequel on veut afficher la valeur. La séquence d'échappement est composé caractère % suivit d'un caractère. *Exemple :*



3.1.1 Format d'affichage

Format d'affichage	Ecriture
Entier Décimal	%d
Entier Octal	%o
Entier Hexadécimal	%x
Entier Non Signé	%u
Caractère	%c
Chaîne de caractères	%s
Flottant	%f
Long Entier	%ld
Long Flottant	%lf
Long Entier non signé	%lu

3.1.2 Code de contrôle

Code de contrôle	Signification
\n	Nouvelle ligne
\a	Bip code ASCII 7
\r	Retour chariot
\b	Espace arrière
\t	Tabulation
\f	Saut de page
\\	Antislash
\ "	Guillemet
\'	Apostrophe
\'0'	Caractère null
\0ddd	Valeur octale (ASCII) ddd
\xdd	Valeur hexadécimale dd

Application de la Fonction affichage : printf. Tapez les programmes suivants :

Programme n°1 :

```
#include<stdio.h>
main()
{
printf("Cours C SUP");
system("pause") ;
}
```

Commentaire(s)

Programme n°2 :

```
main()
{
printf("Cours C SUP");
system("pause") ;
}
```

Commentaire(s)

Programme n°3 :

```
#include<stdio.h>;
main()
{
printf("Cours C SUP");
system("pause") ;
}
```

Commentaire(s)

Programme n°4 :

```
#include<stdio.h>
main()
{
printf("Cours C SUP")
system("pause") ;
}
```

Commentaire(s)

Programme n°5 :

```
#include<stdio.h>
main ;
{
printf("Cours C SUP");
system("pause") ;
}
```

Commentaire(s)

Programme n°6 :

```
#include<stdio.h>
main()
{
int a,b=5 ;
char d="r " ;
printf("Le nombre est %d et %d\n",a,b);
printf("La lettre est %c \n",d);
system("pause") ;
}
```

Commentaire(s) :
Faire la même chose avec des chiffres réels.

3.2 Fonction de Saisie : scanf

Elle permet de saisir des valeurs de variables formatées à partir du clavier. Comme printf elle est composée d'un format et des identificateurs de variables à saisir. A la différence de printf, le format ne peut contenir de texte, il est juste composé du format des valeurs à saisir.

Syntaxe : **scanf (" Format ", &identificateur1,……, &identificateurn) ;**

Remarque:

- Le symbole & est obligatoire devant les identificateurs car scanf attend des adresses et non des valeurs.
- Important: Au début d'un programme, utilisant les fonctions d'affichages et de saisies, il est nécessaire d'écrire #include <stdio.h>, car toutes les fonctions sont déclarées dans ce fichier d'en-tête.

Application de la Fonction affichage : printf. Tapez le programme suivant :

Programme n°1 :

```
#include<stdio.h>
main()
{
int a,b=5 ;
char d="r " ;
printf("Ecrire le chiffre pour a=%d\n") ;
scanf("%d",&a) ;
printf("Le nombre est %d et %d\n",a,b);
printf("La lettre est %c \n",d);
system("pause") ;
}
```

Commentaire(s)

4 L'opérateur de conversion de type.

Il existe deux conversions possibles:

· La conversion implicite. Elle est effectuée pour évaluer le même type de données lors d'évaluation d'expressions. Les conversions systématiques de char en int et en float en double, la conversion se fait toujours du type le plus petit vers le plus long (exemple: de char vers double).

· La conversion explicite. On peut changer le type d'une variable vers un autre type en utilisant l'opérateur cast (type) en le mettant devant l'identificateur de variable à convertir.

Exemple:

```
#include<stdio.h>
/*Déclaration de variable*/
char car;
int a,b,c;
float g;
/* programme principal*/
main()
{
a=4;
b=7;
c=0x41; /*Code Ascii 'A'*/
/*Conversion implicite de a et b en float*/
g=(a+b)/100;
printf("g=%f\n",g);
/*Conversion explicite c en char*/
car=(char) c+3;
/*c est de type entier et sera converti en char*/
printf("car=%c\n",car);
system("pause") ;
}
```

Commentaire(s)

Une transformation peut être forcée par un **cast** :

float x ; int a=5 ; x=(float)a ; / x vaudra 5.0 */*

float x=5.6 ; int a ; a=(int)x ; / a vaudra 5*/*

5 Quelques exercices sur le chapitre 1 & 2 :

Exercice n° 1 :

Soit le programme suivant :

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>

//Déclaration des variables
int heures, minutes, secondes;

/*************************************************************************/
/*                                                                       */
/*              print_heure permet d'afficher l'heure                    */
/*                                                                       */
/*************************************************************************/
void print_heures()
{
printf("Il est %d heure",heures);
if(heures > 1) printf ("s");
printf(" %d minute", minutes);
if(minutes > 1) printf ("s");
printf(" %d seconde",secondes);
if(secondes > 1) printf ("s");
printf("\n");
}

/*************************************************************************/
/*                                                                       */
/*              set_heure met l'heure à une certaine valeur              */
/*                                                                       */
/*************************************************************************/
void set_heures(int h,int m,int s)
{
heures=h; minutes=m;secondes=s;
}

/*************************************************************************/
/*                                                                       */
/*              tick met l'heure à une certaine valeur                   */
/*                                                                       */
/*************************************************************************/
void tick()
{
secondes=secondes+1;
if (secondes>=60)
```

```
    {
    secondes=0;
    minutes=minutes+1;
    if (minutes>=60)
      {
      minutes=0;
      heures=heures+1;
      if (heures>=24) heures=0;
      }
    }
}

/*****************************************************************************/
/*                                                                           */
/*                main Programme principal                                   */
/*                                                                           */
/*****************************************************************************/
int main()
{
 int s=0;
 for (s=1;s<=10;s++)
        {
        set_heures(3,32,10);
        tick();
        print_heures();

        set_heures(1,32,59);
        tick();
        print_heures();

        set_heures(3,59,59);
        tick();
        print_heures();

        set_heures(23,59,59);
        tick();
        print_heures();
        }
     system("PAUSE");
     return 0;
}
```

1. Déterminé quels sont le(s) fonction(s) de ce programme ?
2. Donné le(s) nom(s) du programme principal ?
3. Quelles sont les variables globales et les variables locales ?
4. Quels sont les zones commentaires de ce programme ?

Exercice n°2 : Opérateurs d'incrémentation.

Donnez les valeurs prises par les variables durant l'exécution du programme.

```
#include <stdio.h>
//Declaration des variables
int a,b;
main()
{
/* Initialisation des variables */
a=10 ;
b=100 ;
printf("a = %d \n",a++);          //Affiche...
printf("b = %d \n",--b);          //Affiche...
printf("a = %d \n",a);            //Affiche...
printf("b = %d \n",b++);          //Affiche...
printf("a = %d \n",--a);          //Affiche...
printf("b = %d \n",b);            //Affiche...
}
```

Valeurs finales : a =
b =

Exercice n°3 : Opérateurs binaires.

Donnez les valeurs des variables affichées (dans le format précisé) :

```
#include <stdio.h>
unsigned char porta,i;
main()
{
  porta=0x23;              //Initialisation du porta
  i=porta & 0xF0;
  printf("i(hex) = %x\n\n",i);          //Affiche...
  i=porta | 0x10;
  printf("i(hex) = %x\n\n",i);          //Affiche...
  i=~porta;
  printf("i(hex) = %x\n\n",i);          //Affiche...
  i=porta>>1;
  printf("i(hex) = %x\n\n",i);          //Affiche...
  i=porta<<4;
  printf("i(hex) = %x\n\n",i);          //Affiche...
}
```

Exercice n° 4 :Opérateurs composés

Donnez les valeurs des variables affichées (dans le format précisé) :

```
#include <stdio.h>
int i;
int main()
{
        i=2;
        i+=3;
        printf("i = %d\n\n",i);        //Affiche...
        i*=2;
        printf("i = %d\n\n",i);        //Affiche...
        i%=3;
        printf("i = %d\n\n",i);        //Affiche...
        i<<=2;
        printf("i = %d\n\n",i);        //Affiche...
}
```

Exercice n° 5 : Opérateurs relationnels

Donnez les valeurs des variables affichées (dans le format précisé) :

```
#include <stdio.h>
int a,b,c;
int res;
main()
{
        a=2,b=3,c=5;

        res=((a>3)&&(c>5));
        printf("resultat = %d\n\n",res);

        res=((b>2)||(c<4));
        printf("resultat = %d\n\n",res);

        res=!(a<b);
        printf("resultat = %d\n\n",res);

        res=(b!=c);
        printf("resultat = %d\n\n",res);

}
```

Exercice n° 6 :

Donnez les valeurs des variables affichées (dans le format précisé) :
Rem : Le code ASCII de A est 0x61.

```
#include <stdio.h>

int main()
{
    int res;
    char val1,val2;
    val1='a';
    val2='c';
    printf("La valeur1 vaut %c\n\n",val1);        //Affiche...
    printf("La valeur2 vaut %c\n\n",val2);        //Affiche...
    printf("La valeur1 vaut %x\n\n",val1);        //Affiche...
    printf("La valeur2 vaut %x\n\n",val2);        //Affiche...
    val1+=5;
    val2+=2;
    printf("La valeur1 vaut %c\n\n",val1);        //Affiche...
    printf("La valeur2 vaut %c\n\n",val2);        //Affiche...
    res=val1>val2;
    printf("Valeur1 superieure a valeur2(0 Non; 1 Oui): %d\n\n",res); //Affiche…
    return(0) ;
}
```

Exercice n° 7 :

Ecrivez un programme dans lequel on définit une variable (locale) i de type entier, une variable x de type float, une variable y de type double, et une variable c de type char.
Affichez la valeur de ces variables à l'écran. Que constatez vous ?
Affectez une valeur à ces variables et affichez les de nouveau. Que constatez vous ?
Incrémentez chaque variable de 1. Que constatez vous ?
Affectez à i 1.1, à x la valeur Z, et à c la valeur 3.12.Interprétez ?

Chapitre 3: Structures de contrôle. Verif exo

1 Branchement conditionnel :

1.1 Si … Alors… :

```
if (condition) instruction ;
                ou
if (condition)
        {instruction 1 ;
        …
         instruction n ;}
```

Syntaxe :

instruction est une instruction simple (terminé par ;) ou composé (délimité par{ }).Elle sera exécuté si et seulement si la **condition** est vraie.

Application :

Programme n°1 :

```
#include<stdio.h>
main()
{
int a=4,b=5 ;
if (a>b) printf("A>B") ;
printf("fin du programme") ;
system("pause");
}
```

Commentaire(s) :
Faire la même chose en inversant les valeurs de a et b.

1.2 Si … Alors… Sinon :

Syntaxe :

instruction1 sera exécutée si et seulement si la **condition** est vraie ; si elle est fausse on exécutera **instruction2**.

De même, on exécutera le **bloc1** si **condition** est vrai (1), et **bloc2** si **condition** est fausse (0).

```
if (condition) instruction1; else instruction2;
                ou
if (condition)
        {instruction 1 ;          //bloc1
        …
        instruction n ;}
else
        {instruction 1 ;          //bloc2
        …
        instruction n ;}
```

Application :

Programme n°1 :

```
#include<stdio.h>
main()
{
int a=4,b=5 ;
if (a>b) printf("A>B \n") ;
    else printf("A<B \n") ;
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :
Faire la même chose en inversant les valeurs de a et b.

1.3 Si … Alors… Sinon Si :

Syntaxe :

instruction1 sera exécutée si et seulement si **condition1** est vraie ; si elle est fausse on teste la **condition2** …

De même, on exécutera le **bloc1** si **condition1** est vrai (1). Si elle est fausse (0) on teste **condition2**…

```
if (condition1) instruction1;
else if (condition2) instruction2;
...
else if (condition n) instruction n;
                    ou
if (condition1)
        {instruction 1 ;          //bloc1
        …
        }
else if (condition2)
        {instruction 2 ;          //bloc2
        …
        }
  …
else if (condition n)
        {instruction n ;          //blocn
        …
        }
```

Application :

Programme n°1 :

```
#include<stdio.h>
main()
{
int a=4,b=5 ;
if (a>b) printf("A>B \n") ;
   else printf("A<B \n") ;
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :
Faire varier les valeurs de a, b et c.

2 Boucles :

2.1 Tant que… Faire…:

Syntaxe :

while (condition) instruction;
ou
while (condition)
{instruction1 ;
…
instruction n ;
}

On répète le(s) **instructions**(s) tant que la **condition** est vraie.

Application :

Programme n°1 :

```
#include<stdio.h>
main()
{
int b=0 ;
while (b<5)
        {
        b=b+1;
        }
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :
Afficher les valeurs de b dans la boucle while et en dehors.

2.2 Faire… Tant que…:

Syntaxe :

On répète le(s) **instructions**(s) tant que la **condition** est vraie.

```
do instruction; while (condition);
                ou
do
        {instruction1 ;
        …
        instruction n ;
        }
while (condition) ;
```

Application :

Programme n°1 :

```
#include<stdio.h>
main()
{
int b=0 ;
        do
        {
        b=b+1;
        } while (b<5);
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) : Afficher les valeurs de b dans la boucle do while et en dehors.

2.3 Faire… jusqu'à…:

Syntaxe :

```
for (expr1 ; expr2 ; expr3)
        {instruction1 ;
        …
        instruction n ;
        }
```

expr1 initialise l'expression avant l'entrée dans la boucle.
expr2 : tant que cette expression est vraie, la boucle continue.
expr3 représente l'incrémentation de la variable testée.

Cette boucle est idéale dans le cas où on sait d'avance combien de fois on vas la parcourir.

Exemple : *Affichage des 26 lettres de l'alphabet.*

```
char car;
for(car='A';car!='Z'+1;car++) printf("%c",car);
```

Application :

Programme n°1 :

```c
#include<stdio.h>
main()
{
int b=0,i;
        for(i=0;b<5;i++)
        {
        b=b+i;
        }
printf("fin du programme \n") ;
system("pause");
}
```

```c
#include<stdio.h>
main()
{
int b=0,i;
        for(i=0;b<5;++i)
        {
        b=b+i;
        }
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :
Afficher les valeurs de b et i dans la boucle for.

Donner la différence entre les deux programmes.

3 Instructions de contrôle :

3.1. Instruction de sortie : break

Cette instruction permet de sortir du corps d'une boucle sans réaliser l'évaluation de l'expression .

Application :

Programme n°1 :

```c
#include<stdio.h>
main()
{
int b=0 ;
        do
        {
        b=b+1;break;
        } while (b<5) ;

printf("La valeur de B=%d",b) ;
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :

3.2. Instruction continue :

Cette instruction permet de sauter toutes les instructions qui suivent et de venir directement à l'évaluation de l'expression.

Application : **Affiche les nombres de 1 à 10 et de 20 à 30.**

Programme n°1 :

```
#include<stdio.h>
main()
{
    int i;
    for(i=1;i<=30;i++)
        {if ((i>10)&&(i<20))continue;
        printf("%d\n",i);
        }
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :

3.3. Instruction de branchement goto :

Cette instruction permet de sauter à un point quelconque du programme signalé par une étiquette (label) suivie de :
Application : **Affiche les nombres de 1 à 10 et de 20 à 30.**

Programme n°1 :

```
#include<stdio.h>
main()
{
    int i;
debut : D;
    for(i=1;i<=30;i++)
        {if ((i>10)&&(i<20))continue;
        printf("%d\n",i);
        }
goto debut ;
printf("fin du programme \n") ;
system("pause");
}
```

Commentaire(s) :

Remarque: ***Afin d'améliorer la lisibilité des programmes, il est vivement déconseillé d'utiliser cette instruction.***

4 Structure de choix : switch

Cette instruction permet, en fonction de différentes valeurs d'une variable de choisir entre plusieurs actions. Si aucune valeur n'est trouvée, ceux sont les instructions situées après default qui seront exécutées.

Syntaxe

```
switch (variable)
        {case constante1 : instruction1 ;
        …
        case constanten : instruction n ;
        default : instruction n+1 ;
        }
```

Application : **Créer un menu.**

Programme n°1 :

```
#include <stdio.h>

main()
{
    int i;
// Présentation du menu
    printf("\n\n\tafficher le mode de démarrage de UC:\n");
    printf("\tSous LINUX  (1):\n");
    printf("\t WINDOWS XP (2):\n");
    printf("\t WINDOWS NT (3):\n\n");
    printf("\tSaisissez Votre Choix: ");
    scanf("%d",&i);

    switch(i)
    {
        case 1:{ printf("\n\n\t Le système démarre sous LINUX \n");break;}
        case 2:{ printf("\n\n\t Le système démarre sous WINDOWS XP \n");break;}
        case 3:{ printf("\n\n\t Le système démarre sous WINDOWS NT \n ");break;}
        default: printf("\n\nVotre choix n'est pas valide.\n\n");
    }

    system("PAUSE");
}
```

Commentaire(s) :
Retirer BREAK du programme et tapez 2.

5 Quelques Exercices :

Exercice n°1 : Boucle.

Réalisez un programme qui compte de 1 à 1000 (à l'aide d'un boucle for) et affiche tous les multiples de n. n (entier positif) sera saisit au clavier préalablement.
Vous utiliserez un test sur l'opérateur modulo.

Exercice n°2 : Calculatrice.

Réalisez un programme qui effectue, au choix de l'utilisateur, l'addition, la soustraction, la multiplication, ou la division de deux nombres réels (float).
Après avoir saisie les deux réels, affichez un menu de choix de l'opération à réaliser. La présentation sera la suivante :

Calculatrice

0 Addition
1 Soustraction
2 Multiplication
3 Division

Entrez votre choix :_
Le résultat de votre opération est :

Chapitre 4: Les Fonctions.

1 Introduction:

Afin de réduire la taille du programme il est souvent préférable de définir sous forme de fonction une même suite d'instructions appelée plus d'une fois dans le programme. La fonction principal d'un programme "C" est définie grâce au mot clé "**main**". Une fonction ou procédure ou encore sous programmes est une suite d'instructions élémentaires, mais vue du programme principal main(), elle représente une seule action. Elle est la base des langages structurés.

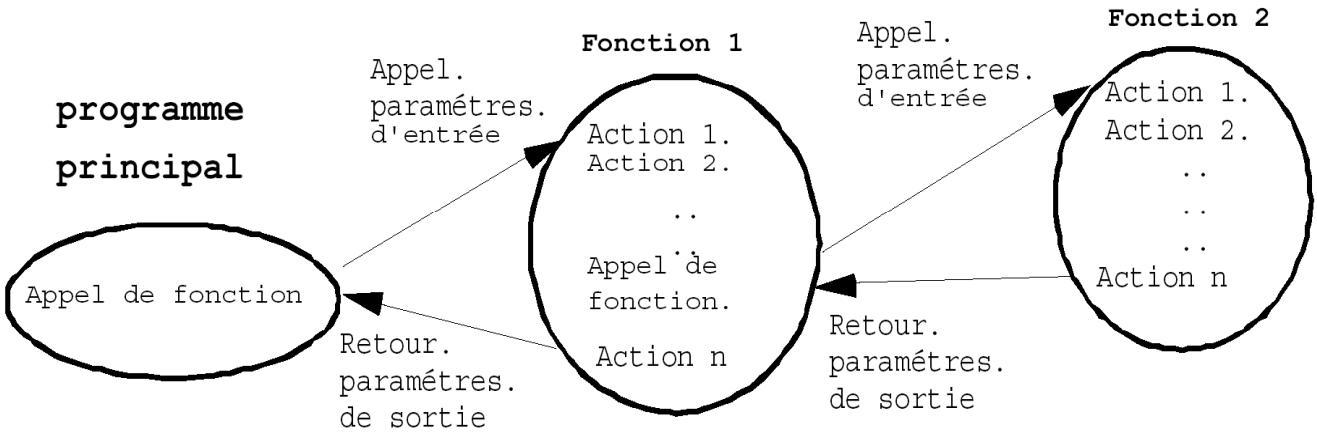
L'écriture de tous programmes commence par définir un algorithme qui décrit l'enchaînement de toutes les actions (fonctions). La réalisation d'un algorithme c'est la décomposition en une suite de fonctions simples pouvant réaliser une ou plusieurs fonctions beaucoup plus compliquées. Un algorithme doit être le plus clair possible.

Une fonction est caractérisée par un appel et un retour.

- L'appel est effectué par le programme principal main() ou une autre fonction.
- Le retour s'effectue après la dernière action de la fonction appelée et le programme continu juste après l'appel de la fonction.

Une fonction peut appelée une autre fonction. Le nom donné à une fonction doit-être représentatif de la fonction réalisée (exemple : perimetre_cercle pour une fonction calculant le périmètre d'un cercle).
Analogie: Pour un travail donné, l'homme décompose toujours en tâches élémentaires avant de réaliser la globalité.

Le programme principal main() et les fonctions ont besoin de se communiquer des valeurs. Lors d'un appel d'une fonction les valeurs passées sont les paramètres d'entrée et à la fin d'une fonction les paramètres renvoyés sont les paramètres de sortie.



2 Déclaration :

2.1 La mise en place d'une fonction dans un programme.

Le compilateur commencera toujours sont exécution au début du programme. La structure d'un programme C complétée par les fonctions est alors :

Exemple :

(1) Déclaration d'une ou plusieurs librairie(s)	#include <stdio.h> #include <math.h>
(4) Déclaration d'une ou plusieurs fonction(s) en faisant un prototypage.	Fonc1() ; Fonc2(char Z) ; Int fonc3(int a,intb);
(2) Déclaration de constantes et des variables globales et les commandes pour le pré-processeur	int i; // une variable globale de type entier

(3) Programme principal	main()
	{
	
	// Le programme
	
	}

(5) Le contenu de la ou les fonction(s).	Fonc1()
	{
	// Le programme de Fonc1()
	}
	
	Fonc2(char Z)
	{
	// Le programme de Fonc2()
	}
	
	Int fonc3(int a,intb);
	{
	// Le programme de Fonc3()
	}
	//etc

2.2 Prototypage :

Le prototypage permet au compilateur de faire une vérification de type au moment de la définition et de l'appel. Il est formellement nécessaire de déclarer le prototype d'une fonction avant que celle-ci soit appelée. La syntaxe est la suivante:

Syntaxe : **type nom_de_fonction (type variable1, type variable2…) ;**

Rem : Usuellement cette déclaration s'effectue avant la fonction main ou dans un fichier d'en tête (include).

Exemple :

Programme n°1 :

```
#include<stdio.h>

int saisie(int A,int B, char H);
autre() ;

main()
{

//le programme principal

}
```

Commentaire(s) :

2.3 Définition d'une fonction sans valeur retournée (Procédure) :

La procédure est une fonction particulière sans type (***void*** signifiant rien). Elle accepte des paramètres d'entrées mais ne retourne rien. Elles servent en générale à structurer un programme. Au lieu d'écrire des milliers de lignes dans le programme principal main() les unes derrières les autres, il faut mieux les rassembler dans des fonctions.

Syntaxe :

```
void  nom_de_fonction (void)
      {…
      }
ou
void  nom_de_fonction (type variable1, type variable2…)
      {…
      }
```

2.4 Appel d'une fonction sans valeur retournée (procédure) :

Il dirige le programme principal ou une autre fonction sur la fonction a exécuter en donnant les variables d'entrées.

```
nom_de_fonction (variable1, variable2…) ;
```

Les arguments de la fonction sont passés par VALEUR ; une copie provisoire de chaque argument est passée à la fonction qui ne peut donc pas modifier l'argument initial d'une fonction appelante.

Exemple :
Programme n°1 :

```c
#include <stdio.h>

/*Déclaration des variables globales*/
int a,b,res;

/*Déclaration des prototypes*/
void saisie(void);
void traitement(int val1, int val2);

/*Programme principal*/
main()
{
     saisie();
     traitement(a,b);
     system("PAUSE");
}

/*Déclaration des fonctions*/
void saisie(void)
{
    printf("Donner les valeurs de a et b : ");
    scanf("%d %d",&a,&b);
}

void traitement(int val1, int val2)
{
    printf("\t\tJe traite\n\n");
    /*traitement*/
    res=val1+val2;
}
```

Commentaire(s) :

Application :

Programme n°1 :
Compléter le programme précédant afin de créer une fonction void affichage(void) qui permet d'afficher le résultat.

```
#include <stdio.h>

/*Déclaration des variables globales*/
int a,b,res;

/*Déclaration des prototypes*/




/*Programme principal*/
main()
{




}

/*Déclaration des fonctions*/
```

Commentaire(s) :

2.5 Définition d'une fonction avec valeur retournée :

Elle décrit l'enchaînement de toutes les instructions permettant de réaliser la fonction. Une variable déclarée dans une fonction est locale à celle-ci et elle n'a aucune existence en dehors de la fonction.

Syntaxe :

```
type nom_de_fonction (type variable1, type variable2…)
        {…
        return (valeur_retournée) ;
        }
```

type : le type de la fonction correspond au type de la valeur qui sera retourné par l'expression
return(valeur) ;
On trouvera donc les types : int, float,…

2.6 Appel d'une fonction :

Il dirige le programme principal ou une autre fonction sur la fonction a exécuter en donnant les variables d'entrées et ou l'affectation de la variable de sortie.

```
var= nom_de_fonction (variable1, variable2…) ;
```

Les arguments de la fonction sont passés par VALEUR ; une copie provisoire de chaque argument est passée à la fonction qui ne peut donc pas modifier l'argument initial d'une fonction appelante.

Remarque pour les chapitres suivants : Si on passe un pointeur (une adresse) à une fonction, celle-ci modifiera ainsi l'argument : c'est un passage par VARIABLE ou par ADRESSE. On obtient le même résultat en passant un tableau à la fonction, puisque les tableaux sont considérés par le compilateur comme un pointeur sur le premier élément du tableau.

Exemple :
Programme n°1 :

```
/*Déclaration des librairies*/
#include <stdio.h>

/*Déclaration des prototypes*/
float cube(float a);

/*Programme principal*/
void main()
{
   float val, res;
         printf ("Entrer une valeur: ");
         scanf ("%f",&val);
         res=cube(val);
         printf("\tLa val au cube vaut : %.2f\n\n",res);

         system("PAUSE");
}


/*Fonction*/
float cube(float a)
{
         int i;
         float cub=1;
         for( i=1 ; i<=3 ; i++) cub*=a;
         return(cub);
}
```

Commentaire(s) :

Application :

Programme n°1 :
Reprendre le programme précédent et passer les variables globales en variables locales afin aboutir au même résultat.

```
#include <stdio.h>

/*Déclaration des prototypes*/




/*Programme principal*/
main()
{
 /*Déclaration des variables locales*/
int a,b,res;









}

/*Déclaration des fonctions*/
```

Commentaire(s) :

3 Quelques Exercices :

1. Réalisez une fonction (***facto()***) qui calcule la fonction factorielle :

$$n ! = \prod_{k=1}^{k=n} k$$

Ex : 1 ! = 1 2 ! = 1*2 3 ! = 1*2*3 … n !=1*2*3*…*n

2. Réalisez une fonction (***taycos()***) qui calcule la fonction cosinus(x) suivant le développement de Taylor à l'ordre n. Les deux paramètres à passer à la fonction seront la valeur de x et l'ordre n du développement :

$$\cos(x)= 1 - \frac{x^2}{2 !} + \frac{x^4}{4 !} + \ldots + \frac{(-1)^n . x^{2n}}{(2.n) !}$$

3. Réalisez une fonction (***taysin()***) qui calcule la fonction sinus(x) suivant le développement de Taylor à l'ordre n. Les deux paramètres à passer à la fonction seront la valeur de x et l'ordre n du développement :

$$\sin(x)=x - \frac{x^3}{3 !} + \frac{x^5}{5 !} + \ldots + \frac{(-1)^n . x^{2n+1}}{(2.n+1) !}$$

4. Réalisez une fonction (***saisie()***) qui réalisera la saisie clavier de l'angle (***x***) et de l'ordre du développement (***n***).

5. Réalisez un programme permettant d'afficher la comparaison de la valeur cosinus et sinus, calculées précédemment par développement de Taylor, avec la fonction *cos(x)* et *sin(x)* fournit par la bibliothèque *<math.h>*.

   Vous utiliserez pour ce programme les fonctions réalisées précédemment .

Chapitre 5: Les Tableaux.

1 Introduction:

Un tableau est une suite de cellules consécutives en mémoire contenant des données de **type identique**. C'est ce qu'on appelle un type homogène. Pour déclaré un tableau, il faut lui définir un type de données et une taille. Le nombre de dimension n'est pas limité (tableau 1 dimension, 2 dimensions (matrice), …, n dimension).

Un indice permet de spécifier l'adresse relative de chaque cellule par rapport au début du tableau. Cet indice(s) doit être un entier. La première cellule aura l'indice 0, la seconde l'indice 1, …, la nième (le dernier) l'indice n-1.

Syntaxe :

<type> <identificateur> <Taille>
type nom_tab[nb max d'éléments]

Exemple :

int tab [10] = {1,2,3,4,5 };//Déclare un tableau de 10 entiers. 5 entiers déclarés et 5 initialisés à 0
float t [50] ; / /Déclare un tableau de 50 réels initialisés à 0

Rem: Les tableaux consomment beaucoup de place en mémoire. Il est important de les dimensionner au plus juste. Dans l'exemple: int tab [10]; => Le compilateur réserve 10 cases mémoires avec chaque case qui a la taille d'un int (soit 2 octets).

Dans un programme :
Déclarer un tableau de 5 entier (en global) et afficher l'adresse et le contenu du tableau.

```
#include <stdio.h>

/*Déclaration du tableau en global*/
int tab[2] ;


/*Programme principal*/
main()
{
 /*Déclaration du tableau en local*/
float t [70] ;
}
```

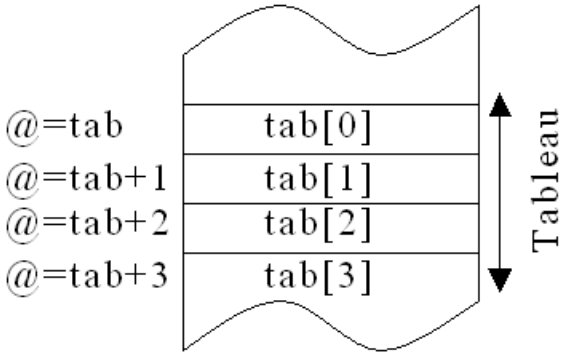
Commentaire(s) :

2 Utilisation des tableaux dans le cadre scientifique :

2.1. Notion d'adresse :

L'identificateur d'un tableau n'a pas pour valeur le contenu du tableau, mais l'adresse du premier élément. Si on se reporte à l'exemple précédent. « tab » correspond à l'adresse du début de tableau.

Donc : tab ou &tab[0] représente l'adresse de tab[0]
tab+1 ou &tab[1] représente l'adresse de tab[1]
tab+2 ou &tab[2] représente l'adresse de tab[2]
tab+3 ou &tab[3] représente l'adresse de tab[3]
…
tab+9 ou &tab[9] représente l'adresse de tab[9]



Application :

Programme n°1 :
Déclarer un tableau de 5 entier (en global) et afficher l'adresse et le contenu du tableau.

```
#include <stdio.h>

/*Déclaration du tableau*/



/*Programme principal*/
main()
{




}
```

Commentaire(s) :

2.2. Initialisation de tableaux :

On peut initialiser les tableaux au moment de leur déclaration.
Exemple : float nombre[4] = {2.67,5.98,-8,0.09};

Nous pouvons être amené à modifier la taille d'un tableau . Il est alors judicieux de le déclarer de le manière suivante.

```
#define  N 5          //Déclare la constante N de valeur 5
int a[N] ;            //Déclare un tableau d'entier de taille N
```

Remarque : **sizeof** (a); //Donne la taille de N*sizeof(int)

Application :

Programme n°1 :

-Afficher les élément du tableau.
-Initialiser le tableau à 999,99.
-Afficher les élément du tableau.
-Afficher la taille du tableau.

```
#include <stdio.h>
#define  N    5              //Déclare la constante N
                             // de valeur 5
/*Déclaration du tableau*/
float nombre[N] = {2.67,5.98,-8,0.09};



/*Programme principal*/
main()
{









}
```

Commentaire(s) :

2.3. Remplissage de tableaux :

Application :

Programme n°1 : Le plus simple est l'affectation élément par élément via l'utilisation d'une boucle.

Par exemple pour copier deux tableaux on fera :

```
#include <stdio.h>

/*Programme principal*/
main()
{
        //Déclaration des deux tableaux
         int i, a[N], b[N]={0,1,2,3,4}
        //Copier le tableau b en a
        for (i=0 ;i<N ;i++) a[i]=b[i];
}
```

Commentaire(s) :

Programme n°2 : On peut également passer par une saisie clavier.

```
#include <stdio.h>

/*Programme principal*/
main()
{
 Printf ("Entrer le %d entier :",N) ;
        for (i=0 ;i<N ;i++) scanf("%d\n",&a[i]); //Saisie les N
                            // valeurs du tableau a[N]
}
```

Commentaire(s) :

2.4. Passage en paramètres à une fonction :

Lorsque l'on passe un tableau en paramètre à une fonction, c'est l'adresse du tableau qui est passé. Il est donc inutile de spécifier sa taille.

Programme n°1 : Le plus simple est l'affectation élément par élément via l'utilisation d'une boucle.

Par exemple pour copier deux tableaux on fera :

```
 //Ce programme permute 2 valeurs du tableau.
// Définition de la fonction.
void echange(double t[], int i, int j)
{
unsigned double tmp;
tmp=t[i];
t[i]=t[j];
t[j]=tmp;
}
```

Commentaire(s) :

2.5. Tableau à plusieurs dimensions :

Le nombre de dimension n'est pas limité.

Syntaxe: **type nom_tab[nb max d'éléments 1ère dim] […] [nb max nième dim]**

On peut se servir d'un tableau avec plusieurs dimensions pour faire des matrices par exemple.

Déclaration :

```
/* Déclaration d'un tableau à 9 éléments - 2 Dimensions */
int tab_2dim[3][3];

/* Le premier élément est tab_2dim[0][0] */
/* Le dernier élément est tab_2dim[2][2] */

main()
{
     ...
}
```



Programme :

```
//Déclaration des librairies
#include <stdio.h>

/*Déclaration des prototypes*/
void raz_mat(int t[][3]);

//Programme principal
main()
{
   int i,j;
   int tab[2][3]={{10,11,12},{13,14,15}};
   printf("Les valeurs sont:\n");
   for(i=0;i<2;i++)
   {
        for(j=0;j<3;j++)
        {
              printf("%d ",tab[i][j]);     //Affichage ligne par ligne
        }
        printf("\n");
   }
   raz_mat(tab);     //Mise à 0 des éléments du tableau appel de la fonction
   printf("Les nouvelles valeurs sont:\n");
   for(i=0;i<2;i++)
   {
        for(j=0;j<3;j++)
        {
              printf("%d ",tab[i][j]);    //Affichage ligne par ligne à 0
        }
        printf("\n");
   }

   printf("\n");
   system("PAUSE");
}

void raz_mat(int t[][3])
{
    int i,j;
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
              t[i][j]=0;
        }
    }
}
```

3 Les chaînes de caractères :

3.1 Déclaration :

Une chaîne de caractère est un tableau de type char, la fin de la chaîne étant indiqué par le caractère '\0'. Leur manipulation est donc identique à celle des tableaux.
Exemple :
On définit un tableau de caractères de 10 éléments. Le message ne pourra contenir que, au maximum, 9 caractères puisque le 10ème est réservé au caractère fin '\0'.

Syntaxe : **char identificateur [nb caractère+1]**

Déclaration:

```
char message[8];          /* Déclaration d'un tableau de chaîne de caractères de 8 éléments - donc 7 caractères*/
```

0	1	2	3	4	5	6	7
'b'	'o'	'n'	'j'	'o'	'u'	'r'	'\0'

3.2 Initialiser des chaînes de caractère :

Initialisation:

```
char message[8];          // Déclaration d'un tableau de chaîne de
                          // caractères de 8 éléments - donc 7 caractères

message[0] = 'b';
message[1] = 'o';
message[2] = 'n';
message[3] = 'j';
    ...
```

- OU déclaration et initialisation en même temps:

```
char message[8] = "bonjour";    // Déclaration d'un tableau de chaîne
                                // de caractères avec « bonjour » initialisé dedans
```

- Pour la modification du contenu, ou la saisie du tableau, on peut utiliser une fonction qui copie une chaîne de caractères dedans: strcpy(tab, "…");

```
char message[8];
strcpy (message, "bonjour");     // Met dans message le mot "bonjour"
```

3.3 L'affichage d'une chaînes de caractère se fait aussi de plusieurs manières :

- Une boucle FOR et un affichage caractère par caractère avec %c:

```
char tab_string1[5]="hello";
for(int i=0;i<5;i++)          // Boucle pour afficher les éléments du tableau
{
        printf("%c",tab_string1[i]);    // Affichage caractère par caractère
 }
```

- Un affichage direct avec %s:

```
char tab_string2[20];
strcpy(tab_string2,"bonjour_les_BM");       // On copie la string
printf("\n\la chaîne de caractère est : %s",tab_string2);        //Affichage
```

La saisie de la chaîne dans le tableau peut se faire caractère par caractère OU avec la fonction : scanf("%s",& tab_string3);

Rem: Ne pas mettre d'espaces et de tabulations (génère des erreurs) – Mettre des « _ »

4 Quelques Exercices :

Les élèves d'une classe obtiennent les notes suivantes :

{8,10,12,16,4,10}

L'instructeur souhaite calculer la moyenne et l'écart type des notes de la classe.

1. Créez une fonction en C permettant de calculer la moyenne de la série de notes (***moy_tab***). Créez par ailleurs le programme principal permettant de tester votre fonction.

Rem : La fonction acceptera comme paramètres d'entrées un tableau (de réels long), ainsi que la taille du tableau (un entier) et retournera comme paramètre de sortie la moyenne de la série (un réel long).

Vous déclarerez une constante N dans le programme principal de valeur 28 pour définir la taille du tableau.

$$\overline{X}=\frac{1}{N}\times\sum_{i=1}^{N}X_i$$

2. Créez une fonction en C permettant de calculer l'écart type de la série de notes (***ecty_tab)***. Modifiez le programme principal précédent pour tester cette nouvelle fonction.

Rem : La fonction acceptera comme paramètres d'entrées un tableau (de réels long), la taille du tableau (un entier), ainsi que la moyenne de la série(un réel long) et retournera comme paramètre de sortie l'écart type de la série (un réel long).

Vous utiliserez la fonction puissance pow, pour calculer le carré, ainsi que la fonction racine carrée sqrt, déclarées dans math.h .

```
double pow (double x, double y);
double sqrt (double x);
```

VARIANCE : $\sigma=\frac{1}{N}\times\sum_{i=1}^{N}(X_i-\overline{X})^2$; ECART-TYPE : $\sqrt{\sigma}$

3. Comme l'instructeur souhaite pouvoir faire les statistiques avec les autres classes, créez une fonction en C permettant d'entrer au clavier les notes (***saisie_tab***). Modifiez le programme principal afin que l'instructeur puissent généraliser son calcul statistique.

Rem : La fonction acceptera comme paramètre d'entrée le nombre de notes (entier) et retournera comme paramètre de sortie le tableau de note (réel long). Le nombre de notes sera saisie au clavier par l'instructeur en début du programme principal.

Chapitre 6: Les Structures.

1 Introduction

Le mot-clé ***struct*** permet de construire des objets hétérogènes. En effet il est habituel en programmation d'avoir besoin d'un mécanisme permettant de grouper un certain nombre de variables de types différents au sein d'une même entité.

On travaille par exemple sur un fichier de personnes et on veut regrouper une variable de type chaîne de caractères pour le nom, une variable de type entier pour le numéro de matricule, etc…

2 Utilisation des structures :

2.1 La mise en place d'une structure dans un programme.

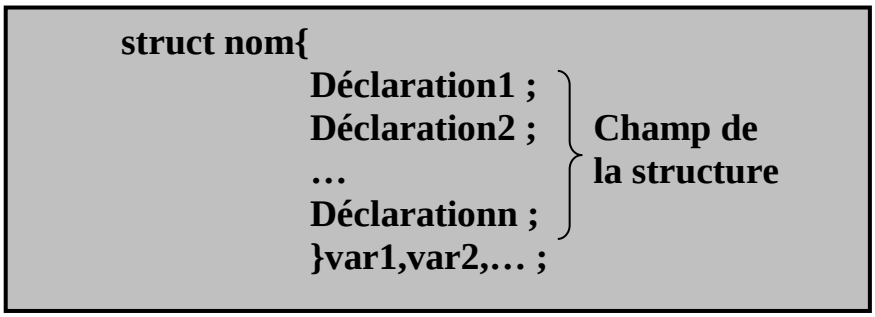
Le compilateur commencera toujours sont exécution au début du programme. La structure d'un programme C complétée par les fonctions est alors :

Exemple :

(1) Déclaration d'une ou plusieurs librairie(s)	#include <stdio.h> #include <math.h>
(6) Déclaration d'une ou plusieurs structure (s).	struct nom{ char pre ; int age ; }N1,N2;
(7) Déclaration d'une ou plusieurs variable de type structure nom.	struct nom N1,N2;
(4) Déclaration d'une ou plusieurs fonction(s) en faisant un prototypage.	Fonc1() ; Fonc2(char Z) ; Int fonc3(int a,intb);

2.2 Déclaration d'une structure :

Syntaxe :



Le nom de la structure est facultatif, ainsi que la liste des variables. Les champs de la structure peuvent être de n'importe quel type.

Exemple:

```
struct fiche {
        int num ;          /* Déclaration du champ num
                              de type entier */
        char nom[20] ;     /* Déclaration du champ nom
                              de type chaîne de caractères */
        char prenom[20] ;  /* Déclaration du champ prénom de type
                              chaîne de caractères */
        int age ;          /*Déclaration du champ age de
                              type entier */
};
struct fiche fic1,fic2 ;   //Déclare 2 variables de type structure fiche
```

2.3 Accès aux champs d'une structure :

Nous avons déclaré deux variables de type structure fic1 et fic2. Pour accéder aux champs de ces variables, nous allons utiliser l'opérateur « . ».
Exemple :

```
fic1.num = 1;              //Déclaration de la 1ère fiche
fic1.nom = "Dupond" ;
fic1.prenom = "Jean" ;
fic1.matricule = 57664728 ;
```

Exemple :

```
fic1.num = 1;                          //Déclaration de la 1ère fiche
strcpy ( fic1.nom, "Dupont" ) ;
strcpy ( fic1. prenom , " Jean " );
fic1.age = 30 ;
```

Rem : On observe avec l'exemple précédent, qu'il faut utiliser la fonction : « **strcpy(var1, var2) ;** » pour copier les chaînes de caractères dans les champs de la structure.

Pour un nombre entier ou réel, une simple affectation au champ de la structure suffit ave le signe =.

2.4 Tableau de structure :

Dans l'exemple suivant, nous allons déclaré trois variables (cercle1, cercle2, cercle3) et un tableau (tab_cercle) de type structure cercle.

La déclaration d'un tableau de structures se fait selon le même modèle que la déclaration d'un tableau dont les éléments sont de types simples.

struct cercle tab_cercle[5] ;

Pour référencer, à la valeur **20**, le rayon du cercle qui à l'index **i** dans ***tab_cercle*** on écrira :

tab_cercle[i].rayon=20 ;

Programme :

```
//Déclaration des librairies
#include <stdio.h>
#include <stdlib.h>

// Déclaration d'une structure : cercle
struct cercle
{
 int x, y;      // Position du centre du cercle
 int rayon;     // Rayon du cercle
};

/* Déclaration de trois structures de type cercle*/
struct cercle cercle1, cercle2, cercle3;

/* Déclaration d'un tableau de 5 structures de type cercle*/
struct cercle tab_cercle[5];

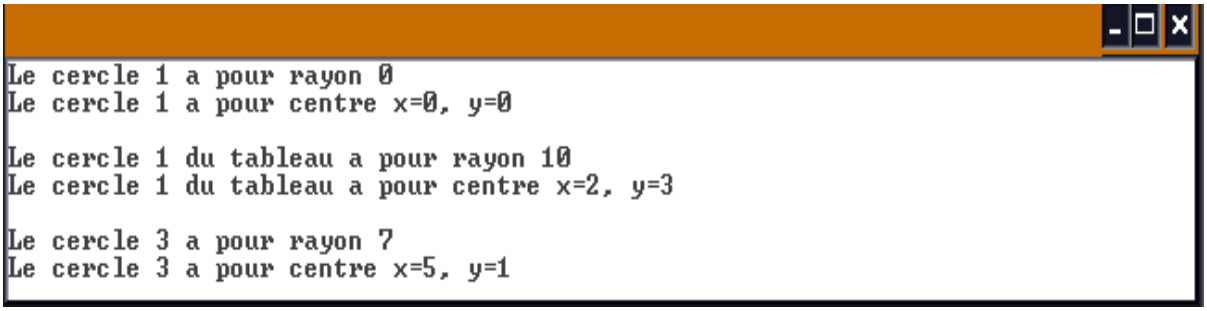
//Programme principal
int main()
{
    /* Initialisation des variables du premier élément (première structure) du tableau de structures*/
    tab_cercle[0].x=2;
    tab_cercle[0].y=3;
    tab_cercle[0].rayon=10;
        /* Initialisation des variables de la structure cercle3*/
    cercle3.x=5;
    cercle3.y=1;
    cercle3.rayon=7;
```

```
    /*Affichage des données du cercle1   (PAS INITIALISEES)*/
    printf("Le cercle 1 a pour rayon %d \n",cercle1.rayon);
    printf("Le cercle 1 a pour centre x=%d, y=%d \n\n",cercle1.x,cercle1.y);

    /* Affichage des données du tab_cercle[0] (INITIALISEES-OK)*/
    printf("Le cercle 1 du tableau a pour rayon %d \n",tab_cercle[0].rayon);
    printf("Le cercle 1 du tableau a pour centre x=%d, y=%d \n\n",tab_cercle[0].x,tab_cercle[0].y);

    /* Affichage des données du cercle3 (INITIALISEES-OK)*/
    printf("Le cercle 3 a pour rayon %d \n",cercle3.rayon);
    printf("Le cercle 3 a pour centre x=%d, y=%d \n\n",cercle3.x,cercle3.y);
    system("PAUSE");
    return 0;
}
```

Le résultat :



3 Les déclarations de type synonyme: *typedef*

On a vu les types de variables utilisés par le langage C: char, int, float ; le chapitre suivant traitera des pointeurs.
Le programmeur a la possibilité de créer ses propres types: **Il suffit de les déclarer en début de programme** (après les déclarations des bibliothèques et les « define ») avec la syntaxe suivante:

Exemples 1 : Créer une équivalence de type.

```
#include <stdio.h>
typedef int entier;              /* on définit un nouveau type
"entier" synonyme de "int" */

entier a,b,c ;    /* On déclare 3 variables de type entier*/

/*Programme principal*/
main()
{
entier d=3 ;

}
```

Commentaire(s) :

entier exprime la même chose que INT.

Exemples 2 : Créer une équivalence de struct.

```
        //Exemple ( programmation d'une entrée sortie sur un port) :
#include <stdio.h>
typedef  struct
        {
        long int adr_port;
        int num_bit;
        } entrée_sortie;   /* déclaration d'un nouveau type entrée_sortie  du type structure */

entrée_sortie RA0={PORTA,0};              /*Déclare et initialise deux variables de type*/
entrée_sortie RB0={PORTB,0};                        /*entrée_sortie*/



void mise_a_un (entrée_sortie es)     /* fonction de mise à 1*/
{                                     /* paramètre d'entrée es de type entrée_sortie */
char c;
c=inportb(es.adr_port);                     /* lit un octet sur le port d'adresse adr_port */
switch(es.num_bit)                          /*Mise à 1 du bit passé en paramètre*/
        {
        case 0 : c=c|0x01;break;
        ....
        ....
        Case 7:c=c|0x80;break;
        }
outportb(es.adr_port,c);              /* copie un octet sur le port d'adresse adr_port */
}
```

Commentaire(s) :

4 Quelques Exercices :

1. Créez une variable ***marin*** de type structure comprenant les champs :

   nom : chaîne de 20 caractères

   prenom : chaîne de 20 caractères

   specialite : chaîne de 20 caractères

   age : entier int non signé

2. Modifiez la déclaration afin de créer un nouveau type ***eleve***.

3. Créez un tableau ***classe*** de taille ***n*** (ici = 30 max) et de type ***eleve***.

4. Créez une fonction permettant de saisir les champs de *n* (max 30) élèves. Le nombre d'élève sera passé en paramètre d'entrées ainsi que le tableau.

5. Créez une fonction permettant d'afficher les *n* (max 30) élèves de la classe. Le nombre d'élève sera passé en paramètre d'entrées ainsi que le tableau.

6. Créez le programme principal. L'utilisateur fixe le nombre d'élèves *n* de la classe et le programme demande les renseignements des *n* élèves, puis les affiche.

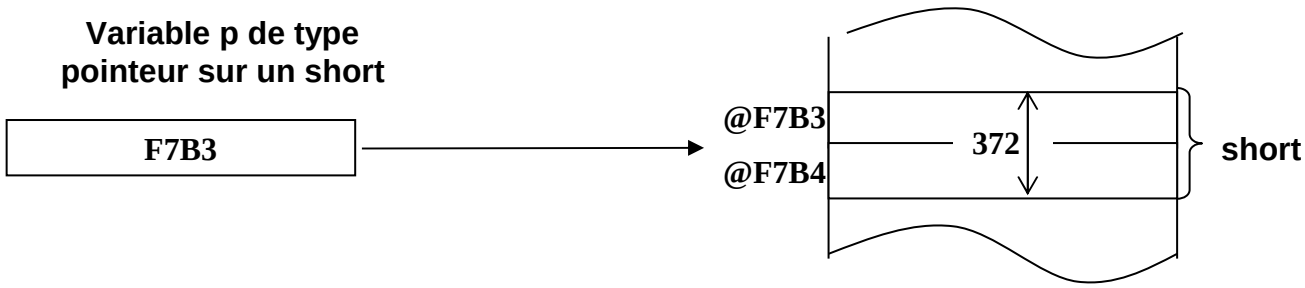
Chapitre 7: Les Pointeurs.

1 Introduction

Un pointeur est une variable qui contient l'adresse d'une autre variable. Selon le type de donnée contenu à l'adresse en question, on aura un **pointeur d'entier, de float, double, char**, ou de tout autre type. En accédant à cette adresse, on peut accéder indirectement à la variable et donc la modifier.

Exemple :

```
short *p ;          //p est un pointeur sur un entier.
```

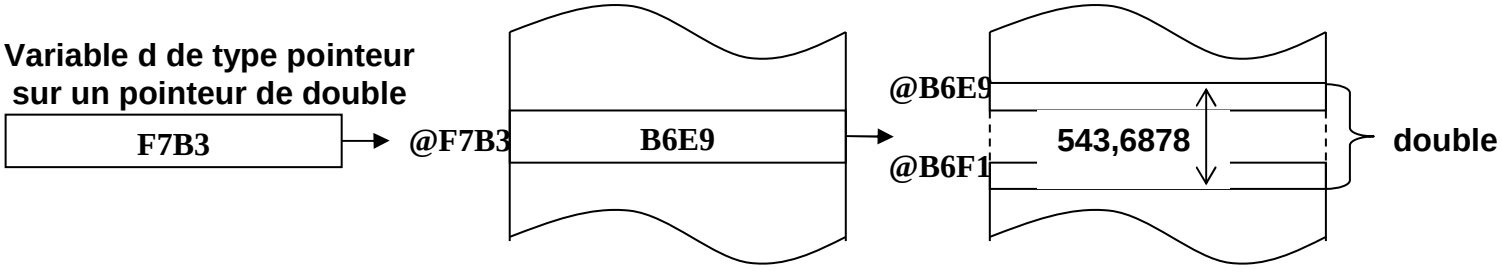


2 Déclaration de pointeur:

Pour déclarer une variable pointeur vers un type de base, on procède comme si l'on déclarait la variable ayant le même type de base et l'on rajoute * devant le nom de la variable.

Exemple :

```
int *i ;            //i est un pointeur sur un entier.
float *f ;     //f est un pointeur sur un réel.
char *c ;      //c est un pointeur sur un caractère.
double **d   //d est un pointeur sur un pointeur de type double.
```



Pour pouvoir accéder à la valeur d'un pointeur il suffit de rajouter une étoile * devant l'identificateur de pointeur. L'étoile devant l'identificateur de pointeur veut dire **valeur pointée par**.

*p renvoit la valeur pointée par p.

Exemple :

Commentaire(s)

```
#include<stdio.h>
main()
{     int *p;
     int i=1234;
//la valeur du pointeur p prend l'@ de i
     p=&i;
//affiche la valeur de la variable p, soit l'@ de i
  printf("%X\n",p);
 //affiche la valeur pointée par p
   printf("%d\n",*p);

}
```

5 Arithmétique sur les pointeurs :

On peut effectuer sur les pointeurs les opérations suivantes:
On peut incrémenter ou décrémenter, ce qui permet de sélectionner tous les éléments d'un tableau. Un pointeur sera toujours incrémenté ou décrémenté du nombre d'octets du type de variable pointée.
Exemple : Un entier occupe deux octets donc un pointeur de type int sera incrémenté ou Décrémenté de deux octets.

```
#include <stdio.h>
/*Déclaration et initialisation d'un tableau d'entier*/
int i[5]={4,-4,5,6,7};

/*Déclarationd'une variable de boucle j*/
int j;

/*Déclaration d'un pointeur d'entier*/
int *ptr_i;

main()
{
/*initialistion de ptr_i sur l'adresse de i*/
ptr_i=i;/*Equivalent à ptr_i=&i[0]*/
```

```
/*Affichage de tous les éléments du tableau*/
for(j=0;j!=5;j++)
   {
   /*affichage des éléments du tableau*/
      printf("i[%d]\n",j,*ptr_i);
      /*Incrémentation du pointeur ptr_ide*/
      /*2 octets car chaque élément de i est*/
      /*un entiers donc deux octets*/
      ptr_i++;
      /*Affichage de la valeur du pointeur i*/
      printf("le pointeur ptr_i=%d\n",ptr_i);
   }
system("pause");
}
```

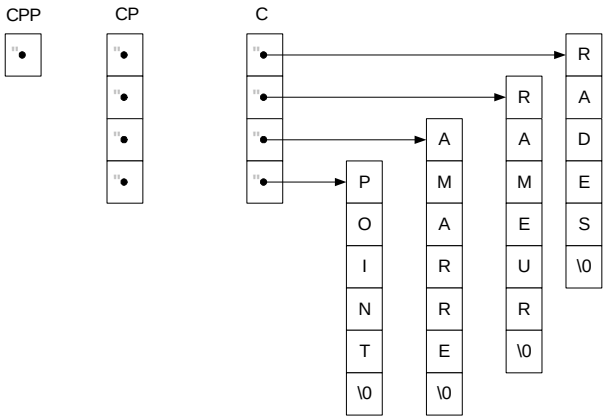
Attention: `*(ptr_mes+5)` **est complétement différent de** `*ptr_mes+5`.
`*(ptr_mes+5)` **c'est la valeur de l'objet pointé par** `ptr_mes+5`.
`*ptr_mes+5` **c'est l'addition de la valeur de l'objet pointé par** `ptr_mes` **et de 5.**

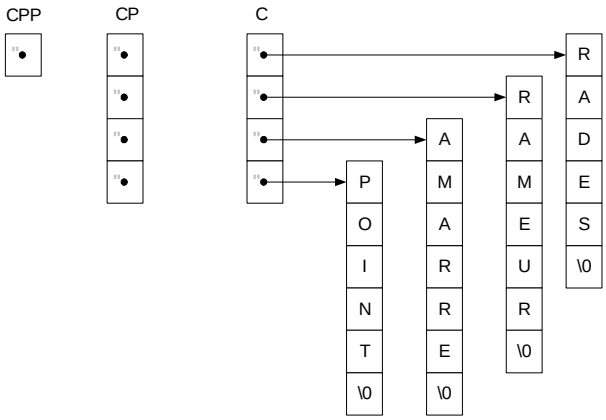
Exercice d'application :

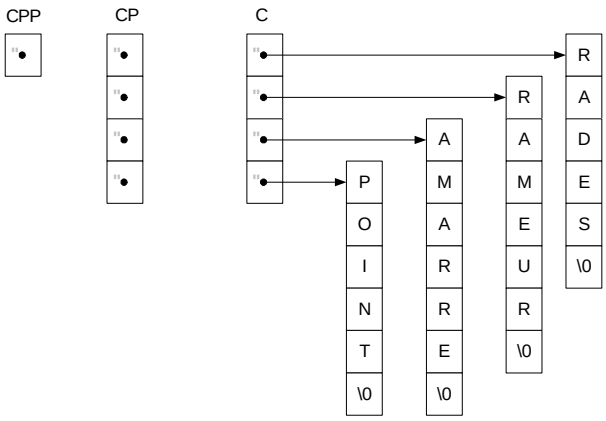
- ✓ A partir du programme précédent établir les liaisons entre le pointeur cpp, le tableau de pointeurs cp et le tableau de pointeurs c par des flèches ?
- ✓ Que va afficher le programme précédent ?

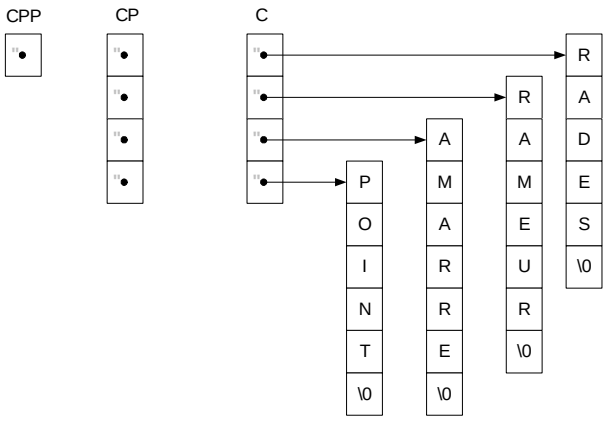
```
#include <stdio.h>
char *C[ ] = {
"rades",
"rameur",
"amarre",
"point" } ;
char **cp[ ]={C+3, C+2, C+1, C} ;
char ***cpp=cp ;

main()
{
printf ("%s", **++cpp+1) ;
printf ("%s",*--*++cpp+2) ;
printf ("%s",*++*--cpp) ;
printf("%s\n",*++*++cpp+3) ;
system("pause");
}
```





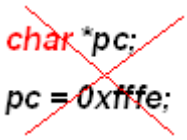




6 Affectation d'une valeur à un pointeur :

Dans les exemples précédents, l'utilisateur ne choisit pas la valeur des adresses mémoire attribuées par le compilateur à chaque variable. L'utilisateur se contente de lire la valeur de ces adresses en l'affichant sur l'écran.

On ne peut pas affecter directement une valeur à un pointeur sauf pour certains compilateurs qui produisent du code machine pour des micro-contrôleurs ou micro-processeurs dans des systèmes autonomes embarqués :
L'écriture suivante est interdite sous DEV C++ :



On peut cependant être amené à définir par programmation la valeur d'une adresse. On utilise pour cela l'opérateur de "cast", jeu de deux parenthèses :

- Par exemple pour adresser un périphérique (adressage physique),
- Par exemple pour contrôler la zone DATA dans un plan mémoire.

Exemples:

```
char *pc;
pc = (char*)0x1000;        // pc est l'adresse 0x1000 et pointe sur un caractère
int *pi;
pi = (int*)0xfffa;         // pi est l'adresse 0xfffa et pointe sur un entier
```

7 Tableaux de pointeur :

Un tableau de pointeurs est un tableau dont les éléments sont des pointeurs.

```
char *mois[13];          // mois : est un tableau de pointeur sur caractère
int *valeurs[5];         // valeurs : est un tableau de pointeur sur entier
```

```
#include <stdio.h>

int *ptr[5];
int i;

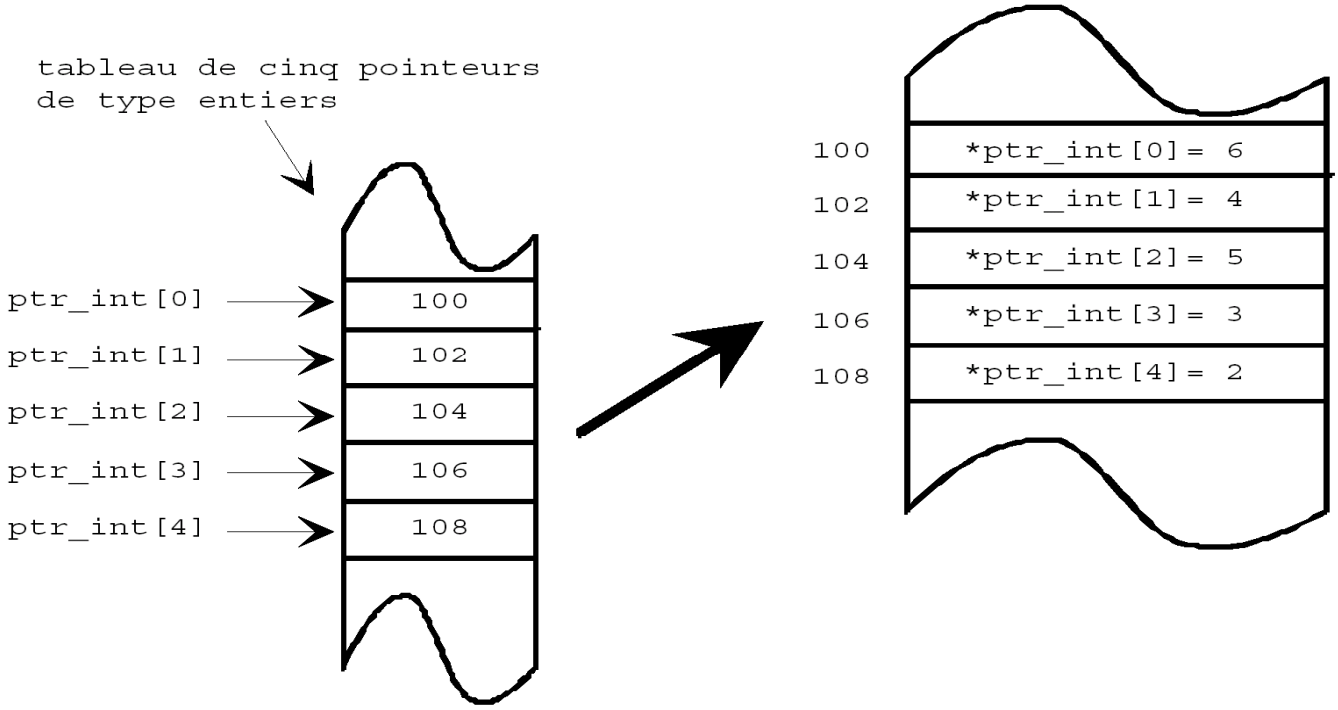
int main()
{
                        /*for(i=0;i!=5;i++) ptr[i]=(int*)malloc(sizeof(int));*/
    *ptr[0]=6;
    *ptr[1]=4;
    *ptr[2]=5;
    *ptr[3]=3;
    *ptr[4]=2;

    for(i=0;i!=5;i++) printf("la valeur pointee par ptr[%d] est %d\n",i,*ptr[i]);

                        /*for(i=0;i!=5;i++) free(ptr[i]);*/
    system("PAUSE");
    return 0;
}
```

Pourquoi le programme ne fonctionnera pas ?

Représentation mémoire:



Le programme a réservé de l'espace mémoire uniquement pour l'adresse de l'élément pointé par ptr_int mais pas pour la valeur pointée par le pointeur ptr_int. Alors pour résoudre ce problème il faut utiliser l'allocation dynamique de la mémoire.
Le principe est de prendre de l'espace mémoire quand on en a besoin et de la rendre disponible lorsque l'on en a plus l'utilité.

Remarque pour les pointeurs :

Si on passe un pointeur (une adresse) à une fonction, celle-ci modifiera ainsi l'argument : c'est un passage par VARIABLE ou par ADRESSE. On obtient le même résultat en passant un tableau à la fonction, puisque les tableaux sont considérés par le compilateur comme un pointeur sur le premier élément du tableau.

8 Allocation dynamique de mémoire

Quand on utilise des pointeurs il est indispensable de toujours savoir quels éléments sont manipulés dans les expressions et si de l'espace mémoire a été réservé par le compilateur pour stocker les objets pointés.
En effet, alors qu'une déclaration de tableau réserve de l'espace mémoire dans l'espace des données pour la variable et ses éléments, celle d'un pointeur ne réserve que la place pour une adresse !

La déclaration : ***double*** *tab[NOMBRE];* - réserve un espace de stockage pour *NOMBRE* éléments de type réel double et la variable *tab* stocke l'adresse de l'élément n° zéro (le premier élément).

Par contre la déclaration : ***double*** **tab;* - ne réserve que le seul espace destiné au pointeur *tab*, AUCUN espace n'est réservé pour une ou plusieurs variables en double précision.

LA RESERVATION D'ESPACE EST A LA CHARGE DU PROGRAMMEUR QUI DOIT LE FAIRE DE FAÇON EXPLICITE, EN UTILISANT LES FONCTIONS STANDARD MALLOC(), CALLOC().

La fonction malloc :

Syntaxe : **Type *malloc(unsigned nombre);**

Malloc retourne un pointeur sur n octets de mémoire non initialisée, ou bien NULL si la demande ne peut pas être satisfaite.
Malloc demande de l'espace mémoire au système d'exploitation, quand il en a besoin de plus il ne sera pas forcément contigu (il garde un lien entre chaque bloque mémoire).

```
char *pc;                    /*pc pointeur sur caractère */
int *pi,*pj,*pk;
float *pr;
pc = (char*)malloc(10);  /* on réserve 10 cases mémoire, soit la place pour 10 caractères */
pi = (int*)malloc(16);  /* on réserve 16 cases mémoire (16 octets), la place pour 4 entiers */
pr = (float*)malloc(24);     /* on réserve 24 places en mémoire, soit la place pour 6 réels */
pj = (int*)malloc(sizeof(int));          /* on réserve la taille d'un entier en mémoire */
pk = (int*)malloc(3*sizeof(int));        /* on réserve la place en mémoire pour 3 entiers */
```

La fonction calloc :

Syntaxe : **Char *calloc(unsigned nombre, unsigned taille);**

Retourne un pointeur sur un espace suffisamment grand pour contenir un tableau de n objets de la taille indiquée, ou bien NULL si la demande ne peut pas être satisfaite.

Exemple :

```
char *s ;
s = (char *) calloc(256, sizeof(char)); /* réserve 256 octets initialisés à 0 */
```

La fonction *free* :

Libération de la mémoire

Syntaxe : **free(le nom du pointeur sur un espace mémoire);**

```
free(pi);          /* on libère la place précédemment réservée pour pi */
```

Free : Libère l'espace mémoire pointé par pi ; elle ne fait rien si pi vaut NULL. pi doit être un pointeur sur un espace mémoire alloué par calloc ou malloc.

9 Les pointeurs et les structures de données

L'avantage d'un pointeur sur une structure :
-Seule l'adresse de la structure est traitée, ce qui vous épargne de devoir passer un exemplaire entier de la structure à la fonction.
-Cela allège considérablement la mémoire.

On peut définir des pointeurs de structures de données.
L'opérateur -> permet d'accéder aux différentes variables de la structure.

```
Syntaxe:
pointeur_d'objet -> identificateur_de_variable = valeur;
                                    ou
valeur = pointeur_d'objet -> identificateur_de_variable;
```

Exemple n°1 :

```c
#include <stdio.h>

// Déclaration d'une structure : cercle
struct cercle
{
 int x, y;      // Position du centre du cercle
 int rayon;     // Rayon du cercle
};

// Déclaration de trois structures de type cercle
typedef struct cercle ptr_cer;

//Prototypage d'une fonction
void ptr_fct(ptr_cer *var_cercle);

//Déclaration lde la fonction
void ptr_fct(ptr_cer *var_cercle)
{
var_cercle->x=10;
var_cercle->y=10;
var_cercle->rayon=10;
}

int main()
{
   ptr_cer *cercle1;
```

```c
      /* Valeurs de départ des variables de la structure cercle1*/
   cercle1->x=99;
   cercle1->y=99;
   cercle1->rayon=99;

   // Affichage des données du cercle1 (PAS INITIALISEES)
   printf("Le cercle 1 a pour rayon %d \n",cercle1->rayon);
   printf("Le cercle 1 a pour centre x=%d, y=%d \n\n",cercle1->x,cercle1->y);

    // Modification des valeurs de cercle1 via une fonction
    ptr_fct(cercle1);

    // Affichage des données avec modif du cercle1
     printf("Le cercle 1 du tableau a pour rayon %d \n",cercle1->rayon);
     printf("Le cercle 1 du tableau a pour centre x=%d, y=%d \n\n",cercle1->x,cercle1->y);

     system("PAUSE");
     return 0;
}
```

Commentaire(s) : On veut changé le ptr_cer *cercle1; par ptr_cer cercle1;
Indiquer les modifications que cela entraîne dans le programme.

Exemple n°2 :

```
#include <stdio.h>

// Déclaration d'une structure : cercle
struct cercle
{
 int x, y;      // Position du centre du cercle
 int rayon;     // Rayon du cercle
};

// Déclaration d'un tableau de 5 structures de type cercle
struct cercle tab_cercle[5];

// Déclaration d'un pointeur de structure de type cercle
struct cercle *ptr_cercle;

int main()
{
    ptr_cercle=&tab_cercle[0];
    // Initialisation des variables du premier élément (première structure) du tableau de structures
    ptr_cercle->x=2;
    ptr_cercle->y=3;
    ptr_cercle->rayon=10;

    // Affichage des données du cercle1    (PAS INITIALISEES)
    printf("Le cercle 1 a pour rayon %d \n",ptr_cercle->rayon);
    printf("Le cercle 1 a pour centre x=%d, y=%d \n\n",ptr_cercle->x,ptr_cercle->y);

    system("PAUSE");
    return 0;
}
```

Application :

Faire la suite du programme pour entrer et afficher les autres valeurs des cercles 2,3,4 et 5.

10 Quelques Exercices :

1°) adr1 et adr2 sont des pointeurs pointant sur des réels. Le contenu de adr1 vaut -45,78; le contenu de adr2 vaut 678,89. Écrire un programme qui affiche les valeurs de adr1, adr2 et de leur contenu.

2°) Créer un tableau d'entier de 10 valeurs {0 à 9} ainsi qu'un pointeur d'entier. Vous afficherez le contenu du tableau en utilisant uniquement le pointeur.

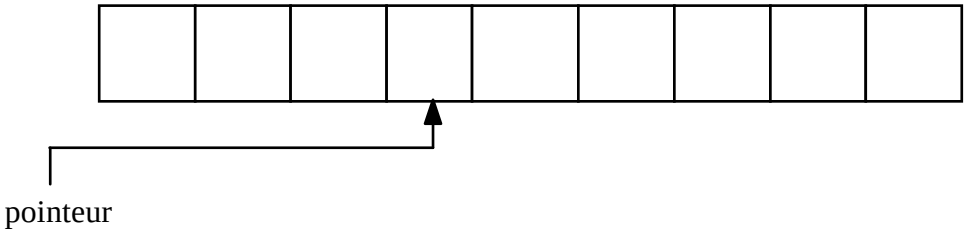
3°) Créez une procédure (*add*) admettant trois paramètres d'entrées a,b,c. Nous souhaitons que le résultat de add soit d'affecter au paramètre c la somme de a et b. Le paramètre c ne peut donc pas être passé par valeur, puisqu'on désire modifier la valeur du paramètre effectif correspondant. Réalisez cette fonction en utilisant un pointeur sur c.

Chapitre 8: Les Fichiers.

1 Introduction

Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).

En C, un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (un char, un int, une structure ...)
Un **pointeur** fournit l'adresse d'une information quelconque.



On distingue généralement deux types d'accès:

Accès séquentiel (possible sur tout support, mais seul possible sur bande magnétique):

- Pas de cellule vide.
- On accède à une cellule quelconque en se déplaçant (via un pointeur), depuis la cellule de départ.
- On ne peut pas détruire une cellule.
- On peut par contre tronquer la fin du fichier.
- On peut ajouter une cellule à la fin.

Accès direct (RANDOM I/O) (Utilisé sur disques, disquettes, CD-ROM où l'accès séquentiel est possible aussi).

- Cellule vide possible.
- On peut directement accéder à une cellule.
- On peut modifier (voir détruire) n'importe quelle cellule.

La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H.

2 Déclaration:

Syntaxe : **FILE *nom_du_fichier ;** */* majuscules obligatoires pour FILE */*

On définit un pointeur. Il s'agit du pointeur représenté sur la figure du début de chapitre. Ce pointeur fournit l'adresse d'une cellule donnée.
La déclaration des fichiers doit figurer AVANT la déclaration des autres variables.

3 Ouverture et fermeture d'un fichier

Il est nécessaire d'ouvrir un fichier avant de travailler dessus puis de le refermer après utilisation. Les fonction ***fopen*** et ***fclose*** de la bibliothèque *STDIO.H* du *C* permettent d'effectuer ces traitements.

3.1 Ouverture d'un fichier: fopen

Lorsqu'on désire accéder à un fichier, il est nécessaire avant tout accès d'ouvrir le fichier à l'aide de la fonction fopen. Cette fonction est déclarée dans la librairie *stdio.h* de la manière suivante :

FILE fopen (char *nom_de_fichier , char *type) ;

Vous l'appellerez en respectant la syntaxe suivante :

fopen (nom_de_fichier , mode) ;

Paramètre d'entrée :

- nom_de_fichier est de type pointeur vers char. La chaîne pointée est le nom du fichier, et éventuellement son chemin, auquel on veut accéder.
- mode est de type pointeur vers char. La chaîne pointée indique le mode d'ouverture du fichier. Elle peut être l'une des chaînes suivantes :

Type	Opération
"r"	Lecture seule.
"w"	Ecriture.
"a"	En ajout écriture à la fin du fichier.
"r+"	Ouverture en lecture et en écriture.
"w+"	Création.
"a+"	En ajout en lecture et en écriture à la fin du fichier.

Il faut en plus dire si un fichier texte "t" ou binaire "b".
Ce qui donne par exemple pour un fichier binaire ouvert en lecture seule le type "rb+".

Valeur retournée :

La fonction ***fopen*** retourne une valeur de type pointeur vers FILE, où FILE est un type prédéfini dans le fichier *stdio.h*. Elle retourne le pointeur de fichier :

- Si l'ouverture à réussi, la valeur retournée permet de repérer le fichier, et devra être passée en paramètre à toute les procédures d'E/S sur le fichier.
- Si l'ouverture s'est avérée impossible ***fopen*** retourne la valeur 0 (NULL).

Exemple :

Une utilisation typique de la fonction ***fopen*** :

```
#include <stdio.h>

// Déclaration du pointeur sur le fichier
    FILE *fp ;

int main()
{
/*Teste si le fichier a bien été ouvert*/
if((fp=fopen("K:\\toto.txt","r"))==NULL)
        printf("Impossible d'ouvrir le fichier!\n");

system("PAUSE");
return 0;
}
```

Une autre façon d'utiliser cette fonction :

```
#include <stdio.h>

// Déclaration du pointeur sur le fichier
    FILE *fp ;

int main()
{
     char *nom="K:\\toto.txt";
     char *attribut="r";

     /*Teste si le fichier a bien été ouvert*/
    if((fp=fopen(nom,attribut))==NULL)
          printf("Impossible d'ouvrir le fichier %s!\n", nom);

system("PAUSE");
return 0;
}
```

3.2 Fermeture d'un fichier: fclose

Quand on a terminé une opération d'entrée sortie sur un fichier il est nécessaire de la fermer. Pour cela on utilise la fonction *fclose*.

La fonction qui gère la fermeture d'un fichier est de la forme suivante :

int fclose (FILE *nom_de_fichier) ;

Vous l'appellerez en respectant la syntaxe suivante :

fclose (nom_de_fichier) ;

Paramètre d'entrée :

Cette fonction admet comme paramètre une valeur de type pointeur vers FILE, qui repère le fichier à fermer.

Valeur retournée :

- En cas d'échec la fonction retourne la valeur -1.
- En cas de succès elle retourne la valeur 0.

Exemples :

Pour fermer le fichier ouvert dans l'exemple précédent:

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
    FILE *fp ;
int main()
{
if (fclose(fp) !=0)
        printf("Erreur dans la fermeture du fichier ! \n");

system("PAUSE");
return 0;
}
```

4 Lecture et écriture sur un fichier :

4.1 Lecture et écriture par caractères :

Fonction fgetc :

La fonction ***fgetc*** lit un caractère dans un fichier. Elle est déclarée de la manière suivante dans la bibliothèque :

```
int fgetc       (FILE *nom_de_fichier) ;
```

Vous l'appellerez en respectant la syntaxe suivante :

```
fgetc   (nom_de_fichier ) ;
```

- **Paramètre :**

nom_de_fichier est un pointeur de type FILE pointant vers le fichier à partir duquel on souhaite lire.

- **Valeur retournée :**

➢ En cas de succès elle retourne la valeur du caractère lu.
➢ Si la fin du fichier est atteint, elle retourne –1 définit comme EOF (End Of File) dans stdio.h.

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_txt ;
int main()
{
//ouverture du fichiera
printf("%c",fgetc(ptr_txt));
//fermeture du fichier
system("PAUSE");
return 0;
}
```

Fonction fputc :

La fonction fputc écrit un caractère dans un fichier. Elle est déclarée de la manière suivante dans la bibliothèque :

```
int fputc      (char car, FILE *nom_de_fichier) ;
```

Vous l'appellerez en respectant la syntaxe suivante :

```
fputc  (caractère, nom_de_fichier) ;
```

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_ecr ;
int main()
{
char f='R' ;
//ouverture du fichier
fputc(f,ptr_ecr) ;
//fermeture du fichier
system("PAUSE");
return 0;
}
```

- **Paramètre :**

caractère est le caractère à écrire dans le fichier.
nom_de_fichier est un pointeur de type FILE pointant vers le fichier sur lequel on souhaite écrire.

- **Valeur retournée :**

➢ En cas de succès elle retourne le caractère écrit.
➢ En cas d'erreur, elle retourne –1 (EOF).

4.2 Lecture et écriture par chaîne de caractères :

Fonction fgets :

La fonction ***fgets*** lit une chaîne de caractères dans un fichier. Elle est déclarée de la manière suivante dans la bibliothèque :

```
char *fgets  (char *s, int n, FILE *nom_de_fichier) ;
```

Vous l'appellerez en respectant la syntaxe suivante :

```
fgets  (chaine, taille,  nom_de_fichier) ;
```

Cette fonction lit une chaîne de caractère s jusqu'au caractère d'interligne \n, et pour un maximum de n-1 caractères, ou jusqu'à la fin du fichier.

- **Paramètre :**

chaîne pointe sur la chaîne de caractère à lire sur le fichier.
taille : taille-1 indique au plus le nombre d'octet à lire. La lecture s'arrête si elle rencontre \n.
nom_de_fichier est un pointeur de type FILE pointant vers le fichier à partir duquel on souhaite lire.

- **Valeur retournée :**

➢ En cas de succès elle retourne la chaîne lue.
➢ En cas d'erreur ou fin de fichier atteinte, elle retourne NULL.

Exemple :

```
#include <stdio.h>
#define N 14
// Déclaration du pointeur sur le fichier
 FILE *ptr_txt ;
main()
{
char tab[N] ;
//ouverture du fichier
fgets(&tab[0],N*sizeof(char),ptr_txt);
//fermeture du fichier
system("PAUSE");
}
```

Fonction fputs :

La fonction ***fputs*** écrit une chaîne de caractères dans un fichier. Elle est déclarée de la manière suivante dans la bibliothèque :

```
int fputs      (const char *s, FILE *nom_de_fichier) ;
```

Vous l'appellerez en respectant la syntaxe suivante :

```
fputs  (chaine, nom_de_fichier) ;
```

- **Paramètre :**

Chaîne pointe sur la chaîne de caractère à écrire sur le fichier.
nom_de_fichier est un pointeur de type FILE pointant vers le fichier sur lequel on souhaite écrire.

- **Valeur retournée :**

➢ En cas de succès elle retourne le dernier caractère écrit. Le pointeur nom_de_fichier avance de la longueur de la chaîne (\0 n'est pas rangé dans le fichier).
➢ En cas d'erreur elle retourne -1.

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_ecr ;
int main()
{
//ouverture du fichier
fputs("bonjour",ptr_ecr) ;
//fermeture du fichier
system("PAUSE");
return 0;
}
```

4.3 Lecture et écriture avec format :

Fonction fprintf :

La fonction ***fprintf*** est plutôt réservée aux fichiers ASCII. Elle permet d'écrire dans un fichier et est déclaré de la manière suivante dans la bibliothèque :

int fprintf (FILE *nom_de_fichier, char format, liste d'expression) ;

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
    FILE *fichier ;
int main()
{ //ouverture du fichier
/* écrit il fait beau  à partir de la position courante du pointeur fichier */
        fprintf(fichier,"%s","il fait beau");
        //il faut définir n
        //fprintf(fichier,%d,n);
        //fprintf(fichier,"%s%d","il fait beau",n);
        //fermeture du fichier
system("PAUSE");
return 0;
}
```

Fonction fsanf :

La fonction ***fscanf*** permet de lire dans un fichier et est déclaré de la manière suivante dans la bibliothèque :

int fscanf (FILE *nom_de_fichier, char format, paramètres) ;

Les paramètres sont des pointeurs. Ils pointent des variables dans lesquelles fscanf déposent les valeurs lues dans nom_de_fichier, après les avoir converties en binaire.

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
    FILE *fichier ;
int main()
{
char tab_char[15] ;
 //ouverture du fichier

        fscanf(fichier,"%s",&tab_char[0]);

//fermeture du fichier
system("PAUSE");
return 0;
}
```

5 Opération en mode binaire :

5.1 Lecture et écriture :

Fonction fread :

int fread (type* destination, n , nombre , FILE *fichier)

Lit sur le fichier indiqué ; nombre objets ayant n octet et les copie les uns à la suite des autres dans l'espace pointé par destination. Elle retourne le nombre d'objets lus.

Exemple :

La fonction fread permet de recopier le contenu d'un fichier dans un tableau.

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_txt ;
char tab[1000]; // permet de stocker jusqu'à 1000 caractères « buffer »

int main()
{
int val ;
//ouverture du fichier

val=fread(tab,sizeof(char),999,ptr_txt);

//affiche le nombre de caractère dans le fichier pointer par ptr_txt.
printf("%d\n",val);
 //affiche le contenu du tab.
printf("%s",tab);

//fermeture du fichier

system("PAUSE");
return 0;
}
```

Fonction fwrite :

int fwrite (type* source, n , nombre , FILE *fichier)

Ecrit sur le fichier indiqué les nombre objets de n octet qui se trouvent les uns à la suite des autres dans l'espace pointé par source. Elle retourne le nombre d'objets écrits.

Exemple :

La fonction fwrite permet de recopier le contenu d'un tableau dans un ficher.

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_txt ,*ptr_ecr;
char tab[1000]; // permet de stocker jusqu'à 1000 caractères « buffer »

int main()
{
int val ;
//ouverture du fichier
val=fread(tab,sizeof(char),999,ptr_txt);
tab[val*sizeof(char)]='\0'; // mise à zero le reste du tableau

//affiche le nombre de caractère dans le fichier pointer par ptr_txt.
printf("%d\n",val);
//affiche le contenu du tab.
printf("%s",tab);

fwrite(tab,sizeof(char),999,ptr_ecr);

//fermeture du fichier
system("PAUSE");
return 0;
}
```

Conclusion les deux fonctions permettent de réaliser la copie de plusieurs fichiers.

5.2 Positionnement dans un fichier :

Fonction fseek :

int fseek (FILE *fichier , long deplacement, int origine);

Positionne le pointeur du fichier associé au fichier indiqué. Une lecture ou écriture ultérieure commencera à partir de la nouvelle position. Celle-ci est obtenue en ajoutant la valeur de déplacement à une certaine base déterminée par l'argument origine qui doit être une des constantes:

. au début du fichier si origine=0
. à la position courante si origine =1
. à la fin du fichier si origine=2

La fonction retourne zéro en cas de succès, une valeur non nulle en cas d'échec.

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_txt ,*ptr_ecr;

int main()
{
int val ;
//ouverture du fichier

// init pointeur au debut du fichier
fseek(ptr_txt,0,0);

//fermeture du fichier
system("PAUSE");
return 0;
}
```

Fonction ftell :

long ftell (FILE *nom_de_fichier)

Retourne la position courante du pointeur de position dans le fichier ou –1 quand il y a une erreur. Le déplacement est mesuré en octet depuis le début du fichier.

Exemple :

```
#include <stdio.h>
// Déclaration du pointeur sur le fichier
 FILE *ptr_ecr ;
int main()
{
//ouverture du fichier
fputs("bonjour",ptr_ecr) ;
printf("le déplacement en octet est de %d \n",ftell(ptr_ecr)) ;
//fermeture du fichier
system("PAUSE");
return 0;
}
```

Fonction feof :

Int feof (FILE *nom_de_fichier)

Si la valeur retournée par cette fonction est différente de 0, la fin du fichier est atteinte
Exemple :

```
#include <stdio.h>
#define N 20
// Déclaration du pointeur sur le fichier
    FILE *fichier ;
int main()
{
       char chaine [N] ;
       while(!feof(fichier))
       //ou while(feof(fichier) !=0)
       {
       fscanf(fichier,"%s",chaine);
       printf("chaîne lue : %s\n",chaine);
       }

system("PAUSE");
return 0;
}
```

6 Exercices

Exercice n°1 :

1.1Réaliser l'ouverture et la fermeture d'un fichier « tonnerre.txt ».

```
fopen (nom_de_fichier , mode) ;
```

```
fclose (nom_de_fichier ) ;
```

1.2 Afficher les caractères contenus dans le fichier avec :

```
fgetc (nom_de_fichier ) ;
```

Essayer avec la fonction suivante :

```
fgets (chaine, taille, nom_de_fichier) ;
```

```
#define LONG 100
char ligne[LONG];
while(fgets(ligne,LONG,ptr_txt)!=NULL);
{
        printf("%s\n",ligne);
}
```

Que constatez vous avec la fonction fgets

1.3 Compter le nombre de lettre présente dans le fichier.

1.3.1 Créez un tableau (tabalph) de type char de 26 emplacements et y mettre les lettre de l'alphabet.

1.3.2 Créez un deuxième tableau (tabcpt) de type char de 26 emplacements initialisé à zéro pour y mettre le nombre d'itération de chaque lettre.

1.3.3 Compter et afficher le nombre de lettre contenu dans le fichier.
Utiliser la fonction fseek pour remettre le pointeur de fichier au point d'origine.

```
int fseek (FILE *fichier , long deplacement, int origine);
```

1.4 Insertion de commentaire dans un fichier

fputc (caractère, nom_de_fichier) ;

1.5 Copier le contenu du fichier dans un autre fichier
Attention créer physiquement un fichier « ecrire.txt ».

Exercice n°2 :

Appliquer les deux types de fonction (fread et fwrite) pour réaliser la même chose que dans l'exercice n°1.