Programmation structurée en langage C

PROGS

Mireille Goud

Sommaire

1. Introduction

Historique

Le langage C a été créé par Dennis Ritchie aux Bell Telephone Laboratories en 1972. Il a été conçu dans le but de développer le système d'exploitation Unix.

En raison de sa puissance et de sa souplesse, l'utilisation du C s'est rapidement répandue au delà des laboratoires Bell. Les programmeurs ont commencé à l'utiliser pour développer toutes sortes de programmes. Rapidement, des organisations diverses ont utilisé leurs propres versions du langage C, et de subtiles différences d'implémentation sont devenues un véritable casse-tête pour les programmeurs. En réponse à ce problème, l'American National Standards Institute (ANSI) a formé un comité en 1983 pour établir une définition standard du C, qui est devenu le C standard ANSI. La plupart des compilateurs C adhèrent à ce standard. Certains compilateurs proposent des fonctionnalités qui ne font pas partie de la norme et les programmeurs ont intérêt à ne pas les utiliser pour garder des programmes portables.

Le nom du langage C vient de son prédécesseur qui était appelé B. Le langage B a été développé par Ken Thompson qui travaillait aussi aux laboratoires Bell.

Utilisation du langage C

Le langage C est utilisé pour des projets aussi variés que des systèmes d'exploitation, des traitements de textes, des graphiques, des tableurs ou même des compilateurs pour d'autres langages.

Il existe un large choix de compilateurs et d'utilitaires.

Le langage C++ est une version améliorée du C. Il a toutes les caractéristiques du langage C avec des fonctions supplémentaires pour la programmation orientée objet.

But du cours

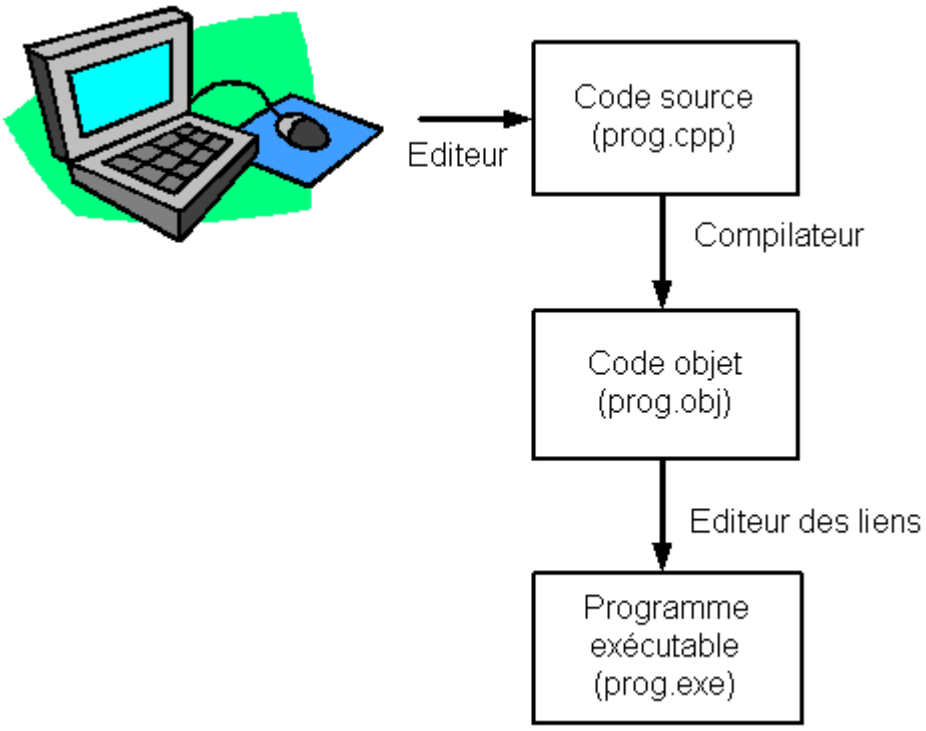
1. Apprentissage d'un langage de programmation.
2. Connaître les fonctionnalités du langage C.
3. Savoir coder un algorithme en langage C et le faire fonctionner.

Développement d'un programme

Pour créer un programme en langage C (ou dans n'importe quel autre langage), il faut suivre les étapes suivantes :

1. Définir les objectifs du programme
2. Écrire le ou les algorithmes de traitement
3. Écrire le programme dans le langage à l'aide d'un éditeur
4. Compiler le programme
5. Exécuter le programme
6. Mise au point du programme tant que l'exécution ne correspond pas aux objectifs :
    a. Corriger le programme à l'aide de l'éditeur,
    b. Retourner au point 3.

Création d'un fichier exécutable



L'éditeur crée un fichier texte (sans caractères spéciaux de mise en forme). Ce fichier texte est soumis au compilateur qui construit un fichier objet. L'éditeur des liens rassemble tous les fichiers objets, les complète par les procédures de librairies et rajoute les liens nécessaires pour former un fichier exécutable.

Utilisation de Visual Studio

Visual Studio permet aussi de créer des programmes en C++. Nous utiliserons seulement les options permettant de travailler en langage C.

Visual Studio utilise la notion de projet, qui correspondra à un répertoire et des fichiers de configuration en plus des fichiers contenant le programme.

Création d'un programme avec Visual Studio :

1. File, New, Projects
   - Choisir l'option : **Win32 Console Application**.
   - Donner un nom au projet (Projectname), exemple : exercice_intro1.
   - Cocher la préférence '**an empty project'** après avoir choisi 'Applications Settings'

2. File, New, Files
   - Afficher la fenêtre 'Solution explorer'
   - Sélectionner 'Source Files' puis bouton de droite, add, New Item.
   - Choisir un nom pour le fichier source (FileName) exemple : salut.
     Le nom du fichier ne comportera pas de point ni d'espace ni de caractères spéciaux
   - Visual C rajoutera le préfixe .cpp au nom du fichier.

3. Saisir le programme :

```
#include <stdio.h>

void main () {
   /*Imprimer un message */
   printf ("Salut, programmeur !\n");
}
```

Comme dans tout éditeur, il ne faut pas oublier d'enregistrer le texte édité.

4. Compiler et créer le fichier exécutable
   - Menu Build : Compile : compile le fichier sans produire l'exécutable.
   - Menu Build : Build salut.exe ou Rebuild all – compile et produit le fichier exécutable.

5. Exécuter
   Cliquer sur ► ou (Ctrl F5), une fenêtre d'exécution DOS s'ouvrira et vous pourrez voir le résultat de votre programme.

Les erreurs de compilation

Si vous enlevez un ' ;', le programme ne se compilera pas : vous aurez une erreur de compilation. Il faudra corriger le programme pour pouvoir l'exécuter. Visual Studio ouvre une fenêtre sous votre fichier source dans laquelle vous avez la liste des erreurs de compilation. Si vous cliquez sur le message d'erreur dans cette nouvelle fenêtre, votre curseur viendra se positionner sur la ligne comportant l'erreur.
Une erreur de compilation apparaît lorsque le compilateur rencontre une erreur de syntaxe dans le programme. Le compilateur vous fournit un message d'erreur et le numéro de la ligne qui contient l'erreur.

Structure de votre premier programme

```
#include <stdio.h>

void main () {
   /* Imprimer un message */
   printf ("Salut, programmeur !\n");
}
```

Insertion d'un fichier avec la directive #include

La directive `#include` indique au compilateur C qu'il doit inclure le contenu d'un fichier dans le programme avant la compilation. Ce fichier contient des instructions en langage C indispensables à votre programme. Les fichiers que vous utiliserez dans ces directives `include`, ont tous une extension .h, ils sont livrés avec votre compilateur et vous ne devez pas les modifier.

La fonction main ()

La fonction `main()` est obligatoire, l'exécution de votre programme débute à la première instruction de cette fonction et se termine à la dernière instruction de main.

Les accolades {}

Les accolades permettent de délimiter les lignes de programmes qui constituent une fonction C. Le début de la fonction commence à l'accolade ouvrante et se termine à l'accolade fermante.

Les instructions

Elles décrivent les actions du programme. Elles affichent les informations à l'écran, elles peuvent lire les données au clavier, effectuent des opérations arithmétiques, etc...
Une instruction est toujours terminée par un point virgule.
Votre premier programme comporte une seule instruction. La fonction `printf` est appelée, elle permet d'écrire un message sur votre écran. Les doubles quotes délimitent la chaîne de caractères qui sera affichée sur l'écran. Les 2 caractères '\n' représente le caractère 'retour chariot' qui provoque un retour à la ligne.

Les commentaires

Ils permettent d'expliquer votre programme aux personnes qui devront le lire pour le modifier ou le corriger. Cette personne peut être vous-même quelques années plus tard et il vous sera très utile d'avoir ces commentaires pour vous rappeler ce que vous avez voulu faire.
Ils commencent par `/*` et se terminent par `*/`. Tous les caractères qui sont compris entre ces 2 séquences ne seront pas interprétés par le compilateur. Un commentaire peut être écrit sur plusieurs lignes.
Un commentaire peut commencer par `//` et il se termine alors en fin de ligne.

En début de programme, mettez systématiquement des commentaires contenant les informaitons suivantes : nom, prénom, date de création du programme et une brève description du programme.

2. Les variables

La mémoire

Un ordinateur utilise de la mémoire vive *(RAM, Random Access Memory)* pour stocker des informations pendant son fonctionnement. La mémoire vive est volatile, elle ne fonctionne que lorsque votre ordinateur est sous tension. La quantité de mémoire installée sur chaque ordinateur est variable. On l'exprime en kilobyte (1 KB = 1024 octets) ou en megabyte (1 MB = 1024 KB). Un octet est égal à 8 bits.

La mémoire RAM est sollicitée de façon séquentielle et chaque octet est identifié par une adresse unique. Cette adresse commence à zéro, pour le premier octet de la mémoire. Un programme travaille avec des données qui sont stockées dans la mémoire vive à une adresse unique pour toute la durée de l'exécution de votre programme.

Les identificateurs

Ils permettent de nommer les objets utilisés dans un programme : variables, constantes, fonctions. Un identificateur doit respecter les règles suivantes :

- il peut contenir des lettres, des chiffres et le caractère '_'
- le premier caractère doit être une lettre. Le caractère '_' peut aussi être la première lettre mais il n'est pas recommandé (parce que ce n'est pas esthétique).
- les lettres majuscules sont différentes des minuscules. Par exemple toto et Toto sont des identificateurs différents.
- la longueur des identificateurs est limitée à un certain nombre de caractères dépendant du compilateur utilisé.
- il ne peut pas être un de ces 33 mots clés réservés pour le langage :

asm	const	else	goto	return	struct	void
auto	continue	enum	if	short	switch	volatile
break	default	extern	int	signed	typedef	while
case	do	float	long	sizeof	union	
char	double	for	register	static	unsigned	

On verra dans les chapitres suivants, l'utilisation de ces mots clés.

Les constantes

Les valeurs numériques sont des constantes (exemples : `123, 45.3`).

Les constantes caractères doivent être entre simple quotte ' (exemples : `'a', 'b', '1', ' ?' ,'$'`).

Les variables

La déclaration de variable va permettre de donner des noms à des zones mémoire. Le programmeur utilisera un nom et le compilateur associera ce nom à une zone mémoire. Le nom d'une variable est un identificateur.

Les types de variables

Le compilateur doit connaître la nature de la variable utilisée dans le programme. Le type d'une variable permet de définir sa représentation en mémoire et les opérations qui pourront lui être appliquées.

Le programmeur choisit le type de la variable selon la nature des informations qui sont traitées dans le programme.

Les types de données en langage C :

Mot clé	Type de variable	Nombre d'octets	Intervalle des valeurs
char	Caractère	1	-128 à 127 ('a' '1' 'x')
int	Entier	4	-2 147 483 648 à 2 147 483 648
short	Entier court	2	-32 768 à 32 767
long	Entier long	4	-2 147 483 648 à 2 147 483 648
unsigned char	Caractère non signé	1	0 à 255
unsigned int	Entier non signé	4	0 à 4 294 967 295
unsigned short	Entier court non signé	2	0 à 65 535
unsigned long	Entier long non signé	4	0 à 4 294 967 295
float	Simple précision virgule flottante	4	$1{,}2 \times 10^{-38}$ à $3{,}4 \times 10^{38}$ (précision 7 chiffres après la virgule)
double	Double précision virgule flottante	8	$2{,}2 \times 10^{-308}$ à $1{,}8 \times 10^{308}$ (précision 19 chiffres après la virgule)

La taille mémoire occupée par les variables de type `int`, `short` et `long` dépend de l'architecture des machines. Le tableau ci-dessus correspond aux tailles utilisées sur les ordinateurs que nous utilisons. La fonction `sizeof` permet de connaître la taille mémoire d'une variable ou d'un type. (Exemple : `sizeof(int)` permet d'obtenir le nombre d'octets réservés pour un entier).
Le type `char` est un entier de 8 bits qui sert à représenter tout les caractères. Chaque caractère est représenté par une valeur numérique.

Les déclarations de variables
Avant d'utiliser une variable dans un programme C, il faut la déclarer. Cette déclaration indiquera au compilateur le nom et le type de la variable. Si votre programme utilise une variable qui n'a pas été déclarée, le compilateur génère un message d'erreur. Une déclaration de variable a la forme suivante :

```
typename varname ;
```

`typename` doit être un des mots clés définissant un type de variable (1ère colonne du tableau précédent) et `varname` est un identificateur qui nommera la variable. On peut déclarer plusieurs variables d'un même type sur la même ligne en les séparant par des virgules.

```
int longueur, largeur ; /* 2 variables entières */
float note, moyenne ;   /* 2 variables à virgule flottante */
char c ;                /* Une variable de type caractère */
```

Le point virgule (;) termine chaque déclaration.

Structure de la fonction main :

```
void main () {
  Déclaration des variables
  Instructions
}
```

Les variables sont déclarées avant d'être utilisées dans les instructions de la fonction.

Initialisation d'une variable

L'opérateur d'affectation est représenté par le signe égal : '='
C'est un opérateur et il engendre une action. Il permet de donner une valeur à une variable.

```
int x ;   /* Declaration de la variable x de type entier */
x = 12 ;  /* Donne la valeur 12 à la variable x          */
```

On peut initialiser une variable lors de sa déclaration :

```
float note = 5.5 ; /* Declare la variable note et l'initialise à 5.5 */
char lettre = 'y' ; /* Variable de type char et initialisation */
```

On ne peut pas initialiser une variable avec une valeur qui ne correspond pas au type déclaré.

```
int toto = 5.8 ; /* warning à la compilation */
```

La valeur 5.8 sera convertie en entier par le compilateur et la variable `toto` aura la valeur 5 après cette initialisation.
Les valeurs introduites dans le code source du programme (12, 5.5, 5.8) sont appelées des **constantes littérales**.

Ecriture de la valeur d'une variable à l'écran

La fonction **printf** permet d'afficher du texte mais aussi les valeurs de variables et d'expressions.

```
printf (Format, liste des variables séparées par des virgules) ;
```

Format : est une chaîne de caractères, soit du texte entre double guillemets, contenant des indicateurs de formats aux emplacements des valeurs de variables à afficher.

Exemple :

```
printf ("Texte à écrire avec format des variables %d %c",
                              varEntiere, varCaractere);
```

Le caractère % indique qu'on va imprimer une valeur et le caractère suivant permet de donner le format de la valeur à imprimer. La liste des variables est composée des noms des variables à afficher dans l'ordre de leur description.

Format	Type de la variable
%d	int
%hd	short
%ld	long
%c	char
%f	float
%Lf	double

Exemples :

```
int jour = 4 ;
int mois = 8 ;
int annee = 1996 ;

printf ("Date de naissance %d / %d / %d \n", jour, mois, annee);
```

Donnera le résultat suivant :

```
Date de naissance 4 / 8 / 1996
```

Lecture de la valeur d'une variable au clavier

La fonction **scanf** permet de lire des valeurs saisies au clavier.

```
n = scanf("Format des variables à lire", liste des adresses de variables);
n sera égal au nombre de variables correctement lues.
```

On utilise les mêmes formats que pour la fonction `printf`, et on utilise l'opérateur & qui permet d'obtenir l'adresse d'une variable. La fonction scanf doit connaître l'adresse de la zone mémoire dans laquelle elle rangera la valeur lue au clavier.

```
float note ;
printf ("Note : "); /* Message affiché sur l'écran pour l'opérateur */
scanf("%f", &note); /* La valeur est saisie et affectée à note  */
```

On peut lire plusieurs valeurs avec une seule instruction `scanf` :

```
int val1, val2 ;
int n ;
n = scanf ("%d %d", &val1, &val2); /* Lecture de 2 valeurs entières */
```

Après cette instruction, les 2 valeurs entières lues au clavier seront affectées respectivement à val1 et val2.

Exercices

2.1 Quel type de variable convient le mieux pour stocker les valeurs suivantes ? Donnez un nom approprié à chaque variable.

Valeur	**Type**	**Nom de la variable**
Âge d'une personne		
Poids d'une personne en kilos		
Prix d'un article en francs		
Votre salaire annuel		
La température		
Distance entre une étoile et la terre		
Nombre d'étudiant dans la classe		
Votre note moyenne en programmation		
Une date de naissance (jour, mois, année)		
Une lettre de l'alphabet		

2.2 Les identificateurs suivants sont-il corrects ?

Identificateur	**Correct**	**NON Correct**
`123partez`		
`x`		
`V001`		
`NB`		
`moyenne_annuelle`		
`moyenneAnnuelle`		
`Prix_en_$`		
`james.bond.007`		
`cette_variable_est_correcte`		
`James-Bond`		

2.3 Écrire un programme qui
- déclare une variable,
- donne une valeur à la variable (affectation) et imprime la valeur de la variable,
- et ensuite lit une valeur au clavier pour l'affecter à cette variable et imprime la nouvelle valeur de la variable :

```
Valeur de la variable : 33
Entrer une valeur : 88 (Cette valeur est saisie au clavier)
La valeur lue est 88
Au revoir
```

2.4 Écrire un programme qui
- déclare 2 variables,
- saisit 2 valeurs au clavier pour initialiser les 2 variables,
- imprime les valeurs des 2 variables
- échange les valeurs des 2 variables
- imprime les valeurs des 2 variables

```
Valeur 1 : 33
Valeur 2 : 55
Valeurs saisies : 33 55
Valeurs après échange : 55 33
```

3. Expressions et opérateurs

Les instructions
Une instruction représente une tâche à accomplir par l'ordinateur. Une instruction est terminée par un ' ;'. En langage C, le format d'écriture des instructions est libre. Ce qui permet de formater un programme en rajoutant des espaces, des tabulations, des retours à la ligne.
Exemple :
```
int toto = 5 ;
```
peut s'écrire aussi :
```
int toto=5 ;
```
mais aussi
```
int
toto
= 5 ;
```
Mais cette dernière présentation n'est pas conseillée car illisible.

Par contre, on ne peut pas rajouter des retours à la ligne dans une chaîne de caractères :
```
printf (
"Salut, programmeur !\n"
); /* La syntaxe est correcte */

printf ("Salut,
programmeur !\n"); /* La syntaxe est INCORRECTE */
```

Les expressions
En langage C, on appelle expression tout ce qui représente une valeur numérique. L'expression la plus simple est constituée d'une seule variable ou d'une constante littérale. Les expressions complexes sont constituées d'expressions plus simples avec des opérateurs.
La valeur d'une expression est la valeur obtenue après évaluation de l'expression.

`2 + 8` : la valeur de cette expression est 10.

L'instruction `x = 8 + 34 ;` évalue l'expression '8 + 34' et attribue le résultat à la variable x.

Les opérateurs
Un opérateur est un symbole qui décrit une opération ou une action à effectuer sur une ou plusieurs opérandes. Un opérande est une expression simple ou complexe. On distingue 4 catégories d'opérateurs :

- l'opérateur d'affectation,
- les opérateurs mathématiques,
- les opérateurs de comparaison,
- les opérateurs logiques.

L'opérateur d'affectation : `variable = expression ;`
L'opérateur d'affectation est le signe égal '='. Attention, l'instruction 'x = y ;' ne signifie pas «x égal à y » mais elle indique au compilateur de copier la valeur de la variable y dans la variable x. L'opérande gauche est toujours le nom d'une variable. Le type de la variable doit être cohérent avec le type de l'expression qui est affectée à la variable.

Les opérateurs arithmétiques binaires

Symbole	Opérateur	Opération	Exemple
+	Addition	Additionne les 2 opérandes	`x + y`
-	Soustraction	Soustrait la valeur de l'opérande droite à la valeur de l'opérande gauche	`x - y`
*	Multiplication	Multiplie les 2 opérandes	`x * y`
/	Division	Divise l'opérande gauche par l'opérande droit	`x / y`
%	Modulo	Donne le reste de la division entière de l'opérande gauche par l'opérande droite	`x % y`

Attention : la division par zéro n'est pas possible. L'opérande droit des opérateurs / et % doit être différent de 0.
La division est une division entière dans une expression comportant seulement des entiers.

Exemples :

```
int x, y ;
x = 10 ;
y = x * 2 ; /* y prend la valeur 20 */
y = x % 4   /* y prend la valeur 2 parce que 10 = 2x4 + 2 */
```

L'opérateur '- 'peut aussi être utilisé en opérateur unaire :

```
x = -y ; /* Change le signe de y */
```

Les opérateurs arithmétiques unaires

Symbole	Opérateur	Opération	Exemple
`++`	Incrémentation	Augmente de 1 la valeur de l'opérande	`x++, ++x`
`--`	Décrémentation	Décrémente de 1 la valeur de l'opérande	`x--, --x`

Ces opérateurs s'appliquent à une variable.
`++x ;` et `x++ ;` sont équivalents à `x = x + 1 ;`
`--x` ; et `x-- ;` sont équivalents à `x = x - 1 ;`
Si l'opérateur unaire est placé devant la variable, la variable est décrémentée ou incrémentée avant l'évaluation de l'expression par contre si l'opérateur est placé après la variable, la variable est modifiée après l'évaluation de l'expression.
Exemples :

```
x = 15 ;
y = x-- ; /* y prend la valeur 15 et x prend la valeur 14 */
x = 10 ;
y = --x ; /* x prend la valeur 9 et y prend la valeur 9 */
```

Hiérarchie des opérateurs et parenthèses
Quand une expression possède plusieurs opérateurs, l'ordre dans lequel les opérations sont effectuées est important :

```
x = 4 + 5 * 3; est équivalent à  x = 4 + (5 * 3);
```

On n'obtiendra pas le même résultat si c'est l'addition qui est réalisée en premier ou si la multiplication est réalisée en premier.

Opérateur	*Priorité*
`++ --`	1
`* / %`	2
`+ -`	3

Dans ce tableau, les opérateurs sont classés par priorité décroissante. Les opérateurs `++` et `--` sont prioritaires sur les autres opérateurs.

L'expression `4 + 5 * 3` vaut `19` car la multiplication est appliquée avant l'addition.

Si une expression contient plusieurs opérateurs de même niveau de priorité, les opérations sont appliquées de **gauche à droite**.
Exemple :

`12 % 5 * 2` vaut 4 car l'opérateur % est appliqué avant l'opérateur * (12 % 5 vaut 2).

Les **parenthèses** permettent de modifier l'ordre d'évaluation des opérateurs. L'expression entre parenthèses sera évaluée en premier.

`(4 + 5) * 3` permet d'additionner 4 et 5 avant de les multiplier par 3.

Une expression peut contenir des parenthèses multiples ou imbriquées. Dans le cas de parenthèses imbriquées, l'évaluation se fait de l'intérieur vers l'extérieur. L'expression :

```
x = (32 - 8) / (2 * (9 / (5 - 2)))
```

se calcule dans l'ordre suivant :

```
1. 24 / (2 * ( 9 / 3))
2. 24 / (2 * 3)
3. 24 / 6
```

`4.` l'expression finale `x = 4` attribue la valeur 4 à la variable x.

Il est recommandé d'utiliser les parenthèses dans toutes les expressions complexes.

Exercices

3.1- Écrire un programme qui lit une valeur entière et qui calcule le carré de cette valeur.

```
Entrez une valeur : 5
Le carré de 5 est 25
```

3.2- Écrire un programme qui lit un nombre de secondes au clavier et qui le transforme en heures, minutes et secondes :

```
Entrez un nombre en seconde : 7556
7556s est egal a 2h 5mn 56s
```

Les opérateurs d'affectation composés

Ces opérateurs permettent d'associer une opération mathématique binaire avec une opération d'affectation. C'est une manière raccourcie d'écrire des opérations courantes.

Exemple : `x = x + 5` peut s'écrire `x += 5`

Symbole	Expression	Est équivalente à
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Exercices

3.3 Donner les valeurs de i et j après les instructions suivantes :

```
int i, j;
1. i = 12 * 3 / 4 - 8;
2. i = 9 * 5 - 11 * 4;
3. j = 11;
   i = 2 * (j / 5);
4. i = 2 ;
   i *= 3;
5. i = 10;
   j = 2;
   i *= 3 + j;
```

Types et opérateurs arithmétiques

- Lorsque les 2 opérandes d'un opérateur binaire sont de types différents, ils sont convertis en un type commun. L'opérande du type le plus « étroit » est converti dans le type de l'opérande du type le plus « large».
- Exemple : Dans une opération entre un int et un float, l'opérande de type int est converti en float => le résultat sera du type float.
- Si une variable reçoit une valeur de type plus « large », la valeur est convertie automatiquement mais avec la génération d'un message de type « warning ».
- L'opérateur % (modulo) ne s'applique pas aux variables de type float ou double.

Opérateurs relationnels

Les opérateurs relationnels sont utilisés pour comparer des expressions en posant des questions du type « x est-il plus grand que 100 ? » ou « y est-il égal à 0 ? ». La valeur finale d'une expression qui contient un opérateur de comparaison est un entier qui vaut 1 si l'expression est vraie et 0 si l'expression est fausse. En langage C, la **valeur 0** correspond à **FAUX** et une valeur **différente de 0** est **VRAI**. Le type booléen n'existe pas en langage C. On utilise un entier pour représenter une valeur booléenne.

Valeur de l'expression	***Résultat de l'évaluation***
0	Faux
Différent de 0	Vrai

Symbole	**Opérateur**	**Priorité**
>	Supérieur	1
<	Inférieur	1
>=	Supérieur ou égal	1
<=	Inférieur ou égal	1
!=	Différent	2
==	Egal	2

Il y a aussi des règles de priorité pour les opérateurs relationnels.
L'instruction :
`x == y > z` est équivalente à l'instruction `x == (y > z)`.
`y > z` vaut 0 ou 1 et ce résultat est comparé à la variable `x`.

Les opérateurs relationnels sont moins prioritaires que les opérateurs arithmétiques.
L'expression `10 + 4 > 5` est équivalente à `(10+4) > 5` et vaut 1.

Attention :

- **une erreur très fréquente est d'utiliser le signe '=' au lieu d'utiliser le signe '==' pour comparer 2 expressions.**

Les opérateurs logiques

Les opérateurs logiques de C permettent de vérifier plusieurs comparaisons dans une même question. Par exemple : dans un programme de saisie de notes, si la note est inférieure à 1 **et** supérieure à 6 alors la note n'est pas valide, il y a une erreur de saisie.

Symbole	Opérateur logique	Priorité
`!`	NON	1
`&&`	ET	2
`¦¦`	OU	3

Utilisation des opérateurs logiques :

Expression	Valeur
`!exp`	Vraie (1) si exp est fausse et Faux(0) si exp est vraie
`exp1 && exp2`	Vraie (1) si exp1 et exp2 sont vraies Faux(0) sinon
`exp1 ¦¦ exp2`	Faux (0) si les 2 expressions sont fausses et Vrai (1) sinon

Exemples :

`(x % 2 == 0) && (x > 6)` : l'expression est vraie si x est paire et supérieur à 6 (Donc l'expression est vraie pour tous les nombres pairs supérieurs ou égaux à 8).

`(x == 2) ¦¦ (x == 3)` : l'expression est vraie si x vaut 2 ou 3.

`! (x < 0)` : vraie si `x >= 0`.

Les opérateurs binaires

Ces opérateurs ne fonctionnent qu'avec des entiers. Ils effectuent des opérations binaires bit à bit.

Opérateurs	Description	Exemple
`~`	complément à 1 (unaire)	`~x`
`&`	et	`x & y`
`¦`	ou inclusif	`x ¦ y`
`^`	ou exclusif	`x ^ y`
`>>`	décalage à droite	`x >> 2`
`<<`	décalage à gauche	`x << 4`

Exemples :

`7 & 12` donne 4 car (0111 & 1100 donne 0100)

`8 >> 2` donne 2 (1000 décalé à droite de 2 donne 0010 soit 2)

`8 << 2` donne 32

Attention, il ne faut pas confondre les opérateurs binaires et les opérateurs relationnels.

Ordre d'évaluation des opérateurs en langage C

Opérateurs	Catégorie	Ordre d'évaluation	Priorité
`() [] -> .`	Référence	->	1
`! ~ ++ -- - + & * (cast)`	Unaire	<-	2
`* / %`	Arithmétique	->	3
`+ -`	Arithmétique	->	4
`>> <<`	Décalage	->	5
`< <= > >=`	Relationnel	->	6
`== !=`	Relationnel	->	7
`&`	Binaire	->	8
`^`	Binaire	->	9
`¦`	Binaire	->	10
`&&`	Logique	->	11
`¦¦`	Logique	->	12
`= += -= *= etc...`	Affectation	<-	13
`,`	séquentiel	<-	14

-> Signifie que les opérandes sont évalués de gauche à droite.
<- Signifie que les opérandes sont évalues de droite à gauche.

Ce tableau est utile pour évaluer les expressions complexes, et surtout lorsque des opérandes sont modifiés au cours de l'évaluation d'une expression.

Exercices

3.4 Calcul de la surface d'un triangle : saisir 2 valeurs de type float pour la hauteur et la base du triangle.

```
Hauteur : 4.2
Base : 3.5
La surface du triangle est 7.35.
```

3.6 Donner les résultats de ce programme :

```
#include <stdio.h>

int main() {
   int a, b, c, d ;
   float x, y, z;
   char car;

   a = 5 / 2 * 3;
   b = 15 - 1 - 3 * 2 / 3;
   d = 5;
   c = d++;
   printf (" a : %d, b : %d, c : %d, d : %d\n", a, b, c, d);

   x = 9 / 2;
   y = 9 / 2.0;
   z = 9.0 / 2;
   printf (" x : %f, y : %f, z : %f\n", x, y, z);

   a = 5;
   x = a / 2;
   y = 5;
   z = y / 2;
   b = y / 2;
   y++;
   printf (" x : %f, y : %f, z : %f, b : %d \n", x, y, z, b);

}
```

.

4. Instructions de contrôle

Par défaut, l'ordre d'exécution d'un programme C se fait séquentiellement. L'exécution commence avec la première instruction de la fonction `main` et se poursuit, instruction par instruction jusqu'à la dernière. Le langage C contient de nombreuses instructions de contrôle qui permet de modifier cet ordre séquentiel.

Instruction if

L'instruction `if` évalue une expression et oriente l'exécution du programme en fonction du résultat de cette évaluation. La syntaxe est la suivante :

```
if (expression) {
      instructions ;
}
```

Si le résultat de l'expression est vrai, les instructions sont exécutées. Dans le cas contraire, l'exécution du programme se poursuit avec l'instruction qui suit l'accolade fermante.

Exemple :

```
if (x > y  && y >= 0) {
   x = y ;
}
```

Dans cet exemple, x va garder sa valeur si `x <= y` et y >= 0 sinon x prendra la valeur de y. Lorsque x est inférieur ou égal à y ou y < 0, l'instruction d'affectation n'est pas exécutée. Après la comparaison, c'est l'instruction qui suit l'accolade fermante qui sera exécutée.

La clause else

```
if (expression) {
      instructions1;
} else {
      instructions2;
}
```

Si le résultat de l'expression est vrai, c'est la série d'instructions1 qui sera exécutée sinon c'est la série d'instructions2 qui sera exécutée.

```
if (age < 16) {
   printf ("Mineur");
} else {
   printf ("Majeur");
}
```

Si age est inférieur à 16, on imprimera le mot 'Mineur' sinon on imprimera le mot 'Majeur'.

Les if imbriqués

```
if (expression1) {
   instructions1;
} else if (expression2) {
   instructions2;
} else {
   instructions3;
}
```

Si expression1 est vraie, la série d'instructions1 sera exécutée, sinon l'expression2 sera évaluée et si l'expression2 est vraie alors la série d'instructions2 sera exécutée sinon, si les 2 expressions sont fausses, c'est la série d'instructions3 qui sera exécutée.

Exemple :

```
if (age < 16) {
   printf ("Junior");
} else if (age < 65){
   printf ("Major");
} else {
   printf ("Senior");
}
```

Exercices

4.1 Écrire un programme qui :

- saisit 4 notes (type float de 1 à 6) au clavier
- affiche la moyenne de ces 4 notes
- une appréciation dépendant de la moyenne :

< 4 => Insuffisant , de 4 à 4.5 => Moyen, >4.5 à 5 Bien et >5 Tbien.

Exemples :

```
Entrez 4 notes : 5 4 3.5 5.5
Moyenne : 4.5 - Moyen
```

4.2 Écrire un programme qui lit un entier et affiche

. s'il est multiple de 3
. s'il est multiple de 2
. s'il est divisible par 6

Exemples :

```
Donnez un entier : 9
Il est multiple de 3

Donnez un entier : 12
Il est multiple de 3
Il est pair
Il est divisible par 6
```

4.3 Écrire un programme qui simule une calculatrice avec 4 fonctions (+ - / *) sur des entiers. Le programme lit en entrée : entier opération entier et affiche :
entier1 opération entier2 = résultat

Exemple :

```
Entrez une opération : 2 + 7
2 + 7 = 9
```

4.4 Programme de facturation avec remise :

- Lire un simple prix hors taxes,
- calculer le prix TTC correspondant (TVA 5,5%),
- calculer une remise dont le taux dépend de la valeur TTC : 0% si inférieur à 1000 F, 1% si supérieur ou égal 1000 F et inférieur à 2000 F et 3% si supérieur ou égal à 2000 F et inférieur à 5000 F, et 5 % si supérieur ou égal à 5000 F.
- Imprimer le prix ttc, la remise et le net à payer.

4.5 Saisir 3 nombres entiers au clavier et imprimer le plus grand.

```
Entrer 3 valeurs : 12 3 10
La plus grande valeur est 12
```

4.6 Saisir 3 nombres entiers (dans les variables val1, val2 et val3) et imprimer les dans l'ordre croissant.
(Échanger les valeurs des variables val1, val2 et val3 pour obtenir : val1 < val2 < val3)

```
Entrer 3 valeurs : 12 3 10
Résultat : 3 10 12
```

4.7 Un magasin est ouvert de 9h à 13h et de 16h à 19h. Saisir une heure au clavier et dire si le magasin est ouvert :

```
Entrer une heure : 12
Le magasin est ouvert à 12 h
```

Instruction Switch

L'instruction if permet à un programme d'exécuter différentes instructions selon une condition qui est vraie ou fausse.
L'instruction `switch` gère un aiguillage multiple. Une expression est évaluée et comparée à un ensemble de valeurs (case).

Syntaxe :

```
switch (expression) {
   case valeur 1 :
      instructions1 ;
      [break ;]
   case valeur2 :
      instructions2 ;
      [break ;]
   case valeurn :
      instructionsn ;
      [break ;]
   default :
      instructionsd ;
      [break ;]
}
```

Si la valeur de l'expression est égale à une valeur listée dans les `case` alors le bloc d'instruction associé sera exécuté. Si la valeur de l'expression ne correspond à aucune des valeurs listées alors c'est le bloc d'instructions suivant `default` qui sera exécuté.

L'instruction `break` permet de sortir de l'instruction `switch`. Attention, s'il n'y a pas d'instructions `break`, les instructions sont exécutées en séquence jusqu'à l'accolade fermante de l'instruction `switch`.

Exemple :

```
#include <stdio.h>
void main () {
    char c;
    printf ("Entrer 'o' pour oui et 'n' pour non : ");
    scanf ("%c", &c);
    switch (c) {
        case 'o':
        case 'y':           // le y sera accepté
            printf ("Oui\n");
            break;
        case 'n':
            printf ("Non\n");
            break;
        default :
            printf ("Vous devez entrer 'o' ou 'n'\n");
     }
 }
```

Exercice

4.7 Saisir un caractère et dire si c'est une voyelle ou une consonne.
Fonctions utiles (avec #include<ctype.h> , isalpha(car) vrai si le caractère est une lettre, toupper(car) donne la lettre majuscule correspondant à car si c'est une lettre minuscule et rend le caractère car dans les autres cas)

Boucle for

Syntaxe :

```
for ( initialisation ; condition ; suivant ) {
   instructions
}
```

Description :

Les *instructions* sont répétées tant que la valeur de *condition* est non nulle (vraie).
Avant la première itération, l'instruction *initialisation* est exécutée;
En général, elle sert à initialiser les variables de la boucle.
Après chaque itération de la boucle, l'instruction *suivant* est exécutée.
En général, elle sert à incrémenter le compteur de la boucle.La boucle for est équivalente à la structure suivante :

Exemple :

```
int i, somme ;
/* Fait la somme des 10 premiers nombres entiers */
somme = 0 ;
for (i = 1; i <= 10; i++) {
   somme = somme + i;
}
printf("Somme : %d ", somme);
```

Exercices

4.9 Imprimer tous les multiples de 7 compris entre 1 et 500

4.10 Imprimer les 20 premières puissances de 2 (2^0, 2^1,...2^{19})

4.11 Dessiner un carré d'astérisques avec des espaces au milieu dont la taille des côtés est saisie au clavier :

```
Taille du carré : 5
*****
*   *
*   *
*   *
*****
```

4.12 Écrire un programme qui affiche la table de multiplication des nombres de 1 à 10, sous la forme suivante :

```
    |   1   2   3   4   5   6   7   8   9  10
----|----------------------------------------
  1 |   1   2   3   4   5   6   7   8   9  10
  2 |   2   4   6   8  10  12  14  16  18  20
  3 |   3   6   9  12  15  18  21  24  27  30
  4 |   4   8  12  16  20  24  28  32  36  40
  5 |   5  10  15  20  25  30  35  40  45  50
  6 |   6  12  18  24  30  36  42  48  54  60
  7 |   7  14  21  28  35  42  49  56  63  70
  8 |   8  16  24  32  40  48  56  64  72  80
  9 |   9  18  27  36  45  54  63  72  81  90
 10 |  10  20  30  40  50  60  70  80  90 100
```

4.13 Donner les résultats de ces programmes :

Les déclarations suivantes sont valables pour toutes les boucles :

```
int i, j;
int somme;
```

Programme	Résultat

`somme = 0;`
`for (i = 1; i <= 6 ; i++) {`
`  somme += i;`
`}`
`printf ("%d\n", somme);`	

`somme = 0;`
`for (i = 1; i <= 10 ; i+=2) {`
`  somme += i;`
`}`
`printf ("%d\n", somme);`	

`somme = 0;`
`for (i = 10; i > 0 ; i-=2) {`
`  somme += i;`
`}`
`printf ("%d\n", somme);`	

`for (i = 12 ; i > 1 ; i -= 2) {`
`  printf ("*");`
`}`
`puts ("");`	

`for (i = 0 ; i < 10 ; i+= 2) {`
`  printf ("*");`
`  for (j = 0 ; j < 10 ; j+= 2){`
`  printf ("-");`
`  }`
`  puts ("*");`
`}`	

`for (i = 0 ; i < 5 ; i+= 1) {`
`  for (j = 0 ; j < i ; j+= 1){`
`  printf (" ");`
`  }`
`  puts ("*");`
`}`	

`for (i = 0 ; i <= 5 ; i+= 1) {`
`  for (j = 0 ; j < i ; j+= 1){`
`  printf (" ");`
`  }`
`  printf ("*");`
`  for (j = 0;j < 10-i*2 ; j+= 1){`
`  printf (" ");`
`  }`
`  puts ("*");`
`}`	

Boucle while

Cette instruction permet de répéter une instruction (ou un bloc d'instructions) tant qu'une condition est vraie.

Syntaxe :

```
while ( condition ) {
      instructions
}
```

Description :

les *instructions* sont exécutées de façon répétitive aussi longtemps que le résultat de *expression* est vrai (différent de 0).
expression est évaluée avant chaque exécution des *instructions*

Exemple :

```
/* Saisit des valeurs tant que la valeur saisie est différente de 0 */
/* Affiche la somme et le nombre de valeurs lues                     */
int val, somme;
int i ;
somme = 0 ;          // Initialisation des variables
i = 0 ;
scanf ("%d ", &val) ;  //
while (val != 0) {    // Test de la condition
   somme = somme + val ;
   i = i + 1 ;
   scanf ("%d ", &val) ;
} // Fin de la boucle
printf("Somme %d ", somme);
printf("Nombre de valeurs %d ", i);
```

Exercices

4.14 Écrire un programme qui permet de lire une série de notes comprises entre 1 et 6 et qui affiche ensuite la moyenne de ces notes.
Contrôler que les notes saisies sont correctes.
La fin de la saisie se fera sur la note égale à 0.

Exemple :

```
Entrez une note : 3.3
Entrez une note : 4.4
Entrez une note : 5.2
Entrez une note : 22
ERREUR : entrez une note entre 1 et 6
Entrez une note : 3.7
Entrez une note : 4.9
Entrez une note : 0
La moyenne est :  4.3
```

4.15 Écrire un programme de jeu :
Générer un nombre aléatoire entre 0 et 100 :
Voici les instructions pour générer ce nombre :
Il faudra rajouter les 2 directives suivantes :

```
#include <stdlib.h> /* pour rand() */
#include <time.h>   /* pour stocker l'heure pour srand */
```

Déclarer les variables suivantes :

```
time_t t;           /* Variable qui permet de stocker l'heure */
int solution ;      /* Variable pour stocker la solution
```

Au début de votre programme introduisez les 2 lignes suivantes :

```
srand((unsigned) time(&t)); /* initialiser le générateur de nombre
                                  selon l'heure courante*/
solution = rand() % 101;    /* reste sera toujours entre 0 et 100 */
```

Saisir une réponse jusqu'à ce que la solution soit trouvée.
Dire si la réponse est trop grande ou trop petite à chaque réponse.

Lorsque la réponse est trouvée :
Donner une appréciation selon le nombre d'essais :

```
nb_essais <= 5 "vous avez eu un peu de chance"
nb_essais <= 7 "bravo"
```

Sinon : "`ce score me semble bien minable`"

4.16 Saisir plusieurs lignes de texte (caractères par caractères) et compter le nombre de lignes, le nombre de caractères et le nombre de mots. On considère qu'un mot est une suite de caractères consécutifs qui ne contiennent pas les caractères : '.', ', ', '?' !, ni d'espaces, de tabulations et de retour à la ligne.

```
Entrer votre texte
Bonjour,
Le texte sera termimé
Par le caractère #
Nombre de mots : 8
Nombre de caracteres : 47
Nombre de lignes : 3
```

4.17 Gestion d'un compte en banque : Saisir une commande sous la forme d'un nombre entier valant 1 pour déposer de l'argent, 2 pour retirer de l'argent, 3 pour afficher le solde du compte et 0 pour sortir. Au début du programme, le solde du compte est 0. On ne peut pas avoir un solde négatif.

Exemple :

```
Commande : 3
Solde 0.00
Commande : 1
Montant de votre depot : 1000
Commande : 2
Montant de votre retrait : 500
Commande : 2
Montant de votre retrait : 600
Retrait impossible
Commande : 3
Solde 500.00
Commande : 2
Montant de votre retrait : 200
Commande : 3
Solde 300.00
Commande : 0
Au revoir
```

4.18 Saisir une valeur entière n (>0) et calculer les n premières puissances de 2. Exemple :

```
Entrer une valeur : 10
2**1 : 2          2**2 : 4          2**3 : 8          2**4 : 16
2**5 : 32         2**6 : 64         2**7 : 128        2**8 : 256
2**9 : 512        2**10: 1024
```

4.19 Écrire un programme qui calcule une valeur à la puissance n.

Exemple :

```
Entrer 2 valeurs : 3 4
3**4 = 81
```

4.20 Saisir une valeur en base 2 et la convertir en nombre décimal.

```
Valeur binaire : 1100
Valeur decimal : 12
```

4.21 Saisir une valeur entière et dire si c'est un nombre premier ou non.

```
Valeur : 13
13 est un nombre premier
Valeur : 10
10 n'est pas un nombre premier
```

4.22 Dessiner un carré d'astérisques avec les 2 diagonales et dont la taille des côtés est saisie au clavier :

```
Taille du carré : 7
*******
**   **
* * * *
*  *  *
* * * *
**   **
*******
```

4.23 Imprimer un losange dont la demi-hauteur est saisie au clavier.

```
Demi-hauteur du losange : 4
   *
  * *
 *   *
*     *
 *   *
  * *
   *
```

Remarque : la demi-hauteur comprend la ligne centrale du losange.

5. Les tableaux

5.1 Tableaux à une dimension

Définition

Un tableau est un ensemble d'emplacements mémoire qui portent le même nom et contiennent le même type de données. Chacun de ces emplacements est un élément du tableau.
Exemple d'utilisation : dans un programme qui gère le calcul des températures moyennes mensuelles, il est plus facile de gérer un tableau à 12 éléments plutôt que de gérer 12 noms de variables.
Un tableau à une dimension ne possède qu'un seul index. Un index est le nombre qui permet d'accéder à un élément du tableau.

Déclaration

`typeDesElements nomDuTableau [nombredElements] ;`
`typeDesElements` est le type de chaque élément du tableau.

Exemples :
```
float temperatures [12] ;
```
Le tableau s'appelle `temperatures` et contient 12 éléments de type `float`

```
int notes [100] ;
```
Le tableau s'appelle `notes` et contient 100 éléments de type `int`.

Utilisation

Un élément de tableau s'utilise comme une variable isolée de même type. On utilise un index entre crochets `(nomTableau[index])` pour adresser un élément d'un tableau. Les éléments sont toujours numérotés de 0 à n-1 (n étant la taille du tableau. Le premier élément est accessible avec l'index 0.

Exemples :
Le 3eme élément prend la valeur 20.1 :
```
temperatures[2] = 20.1
```
Le 3eme élément prend la valeur du 4eme élément :
```
temperatures[2] = temperatures[3] ;
```
Initialisation du 2ème élément par une valeur lue au clavier :
```
scanf ("%d", &temperatures[1]) ;
```
Dans cet exemple, c'est l'adresse du 2ème élément (index 1) qui est passée en paramètre de la fonction scanf.

Initialisation d'un tableau par une boucle :
```
int notes [100] ;
int i;
for (i=0 ; i<100 ; i++) {
      notes[i] = 1 ;
}
```
Les 100 éléments du tableau sont initialisés à 1.

Exercices :

5.1 Écrire un programme qui déclare un tableau de 100 entiers et qui les initialise par des valeurs aléatoires comprises entre 0 et 100.

```
#include <stdlib.h> /* pour rand() */
#include <time.h>   /* pour stocker l'heure pour srand */
```

Déclarer la variable suivante :

```
time_t t;           /* Variable qui permet de stocker l'heure */
```

Au début de votre programme initialiser la 1ère valeur aléatoire :

```
srand((unsigned) time(&t)); /* initialiser le générateur de nombre
                               selon l'heure courante*/
val = rand() % 101;    /* générer un nbre aléatoire entre 0 et 100 */
```

Imprimer toutes les valeurs du tableau.

5.2 Écrire un programme qui saisit les dépenses des 12 mois de l'année et qui les stocke dans un tableau à 12 éléments.
Afficher ensuite la liste des dépenses pour chaque mois, la dépense mensuelle moyenne, le total annuel à partir du tableau créé.

5.3 Saisir du texte qui se termine par un point (.) au clavier et donner le nombre d'occurrences de chaque lettre (minuscule ou majuscule) de l'alphabet :

```
Entrez le texte :
Bonjour,
Dans une semaine, nous serons en vacances.
Utilisation des lettres :
a : 4 - n : 8
b : 1 - o : 4
c : 2 - p : 0
d : 1 - q : 0
e : 6 - r : 2
f : 0 - s : 6
g : 0 - t : 0
h : 0 - u : 3
i : 1 - v : 1
j : 1 - w : 0
k : 0 - x : 0
l : 0 - y : 0
m : 1 - z : 0
Autres : 9
```

Indication ;
Utiliser un tableau (alpha) de 26 caractères dans lequel on met toutes les lettres de l'alphabet.
Utiliser un tableau (compteurs) de 26 entiers pour compter le nombre d'occurrences de chaque lettre.
Pour chaque caractère, le rechercher dans le tableau alpha et utiliser l'indice pour incrémenter le compteur correspondant.

5.4 Saisir du texte terminé par un point dans un tableau de caractères et remplacer tous les espaces par des '-'. Imprimer le résultat.
Exemple :

```
Texte : Pierre et le loup.
Resultat : Pierre-et-le-loup.
```

5.5 Saisir N valeurs entières et les stocker dans un tableau (N constante define).
Trier le tableau par ordre croissant de la manière suivante : chercher la plus petite valeur du tableau et l'échanger avec la 1ère valeur, puis faire la même chose avec la 2ème valeur et les valeurs suivantes jusqu'à ce que le tableau soit trié et imprimer les valeurs du tableau.

```
Valeurs : 800 3 77 33 555 0
Tableau trie . 3 33 77 555 800
```

5.7 Recherche d'une valeur dans un tableau non trié :

Initialiser un tableau d'entiers avec des valeurs aléatoires.
Saisir une valeur au clavier et rechercher cette valeur dans le tableau.
Donner la position de la valeur dans le tableau.

Exemple :

```
Tableau : int tab [] = {99, 22, 44, 12, 87, 101, 2} ;

Valeur : 44
44 est à la position 3
Valeur : 101
101 est à la position 7
Valeur : 13
13 n'est pas dans le tableau
```

5.8 Recherche d'une valeur dans un tableau trié :

Initialiser un tableau d'entiers avec des valeurs aléatoires mais en gardant les valeurs du tableau par ordre croissant. Pour chaque nouvelle valeur, il faut chercher son emplacement parmi les valeurs existantes et déplacer les valeurs qui sont à sa droite.
Imprimez le tableau après chaque insertion.
Saisir une valeur entière au clavier et chercher si la valeur est dans le tableau. Si elle est dans le tableau, affichez son indice, sinon imprimez le message : ' Valeur non trouvée'

5.9 Gestion d'une liste d'attente dans un magasin :

Chaque fois qu'un client entre dans le magasin, on lui attribue un numéro qui est incrémenté de 1 par rapport au numéro du client précédent. Le premier client a le numéro 0. Après le numéro 999, on retrouve le numéro 0.

Le magasin peut accueillir seulement 10 personnes à la fois. Les numéros des 10 personnes en attente sont gardés dans un tableau de 10 éléments. Lorsqu'un client est servi, le premier client en attente est appelé et sorti de la liste d'attente. Les clients qui arrivent alors que le magasin est plein ne sont pas acceptés.
Le programme proposera un menu :

1 => arrivée d'un client,
2 => impression du numéro du client à servir et le sortir de la liste.
0 => sortie du programme : autorisée seulement s'il n'y a plus de clients dans le magasin.

5.2 Tableaux à plusieurs dimensions

Définition

Un tableau à plusieurs dimensions possède plusieurs index. Un tableau à n dimensions a n index. En langage C, il n'y a pas de limite au nombre de dimensions d'un tableau.

Utilisation

```
float temperatures [12][31] ;
```

Le tableau s'appelle `temperatures` et contient 372 (12 x 31) éléments de type `float` Il est utilisé pour stocker les températures journalières des 12 mois de l'année. Certains éléments ne seront pas initialisés pour les mois qui ont moins de 31 jours.

```
temperatures [1][5] = 20.1 ; /* temperature du 6 février */

int notesAnnuelles [16][3];
```

Le tableau s'appelle `notesAnnuelles` et contient 48 éléments de type `int`. Il est utilisé pour stocker les 3 notes annuelles des 16 étudiants.

```
int notes [16][10][3];
```

Le tableau s'appelle `notes` et contient toutes les notes des 16 étudiants. Chaque étudiant a reçu 3 notes dans les 10 matières au programme. Ce tableau est formé de 480 éléments de type `int`.

Utiliser des constantes définies par `#define` pour déclarer la taille des tableaux.

Initialisation dynamique d'un tableau :

```
#define NBETUDIANTS 16
#define NBNOTES 3
int notes [NBETUDIANTS][NBNOTES] ;
for (i=0 ; i<NBETUDIANTS ; i++) {
   for (j=0 ; j<NBNOTES ; j++) {
      notes[i][j] = 1 ;/* Les 48 éléments du tableau sont initialises à
1.*/
   }
}
```

Initialisation à la déclaration :

```
int matrices [2][5] = { {1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}} ;
```

Le programme est plus lisible si on regroupe les valeurs entre accolades. Les virgules sont obligatoires même s'il y a des accolades.

Taille d'un tableau

La mémoire occupée par un tableau dépend du nombre et de la taille des éléments qu'il contient. La taille d'un élément dépend de la taille allouée par votre compilateur pour le type de la donnée (ex char = 8 bits ou 1 octet et int = 32 bits ou 4 octets).

La fonction `sizeof (element)` permet de connaître la taille allouée pour un élément. L'instruction `sizeof(notes)` permet de connaître la taille allouée pour le tableau notes.

La taille maximale d'un tableau dépend de la taille mémoire disponible sur votre ordinateur et de la taille occupée par l'ensemble des variables du programme.

Exercices :

5.10 Écrire un programme qui initialise un tableau d'entiers :

```
int table [NBLIGNES][NBCOLONNES] ;
```

avec des valeurs saisies au clavier.
Après l'initialisation du tableau, chercher dans le tableau si un des éléments contient la valeur 0 et afficher le résultat.

Exemple avec `NBLIGNES = 3` et `NBCOLONNES = 4` :

```
Taille du tableau 48 octets
Entrer 1 entier numero 1, table[0][0] : 5
Entrer 1 entier numero 2 , table[0][1]: 6
......etc
Entrer 1 entier numero 12, table[2][3] : 2
La table contient une valeur nulle
```

5.11 Initialiser un tableau d'entiers à 2 dimensions et afficher les valeurs du tableau ainsi que les sommes par ligne et par colonne :

```
Taille du tableau 24 octets
Entrer la valeur no :  1 (table[0][0])4
Entrer la valeur no :  2 (table[0][1])5
Entrer la valeur no :  3 (table[0][2])6
Entrer la valeur no :  4 (table[1][0])1
Entrer la valeur no :  5 (table[1][1])2
Entrer la valeur no :  6 (table[1][2])4

|      4 |      5 |      6 |    15
|--------|--------|--------|-------
|      1 |      2 |      4 |     7
|--------|--------|--------|-------
|      5 |      7 |    10
```

5.12 Soit le tableau notes :

```
float notes [NBETUDIANTS][NBMATIERES][NBNOTES];
```

qui contient pour chaque étudiant toutes les notes attribuées dans chaque matière.
Saisir toutes les valeurs au clavier. Chaque note doit être comprise entre 1 et 6. La note –1 est attribuée aux étudiants absents au test.
Calculer la moyenne par étudiant, dans chaque matière et la moyenne globale.
Indiquer le numéro de l'étudiant qui a la meilleure moyenne annuelle. Les étudiants sont numérotés de 1 à `NBETUDIANTS` selon leur position dans le tableau.

5.13 Soit un tableau d'entiers à 2 dimensions N lignes et N colonnes. Initialiser le tableau statiquement ou avec des valeurs lues au clavier. Vérifier si le tableau est un carré magique : la somme de chaque ligne est égale à la somme de chaque colonne et à la somme de chaque diagonale.

6. Utilisation de pointeurs

Définition

La mémoire vive est constituée de milliers d'emplacement mémoire rangés de façon séquentielle. Chaque emplacement a une adresse unique, comprise entre 0 et une valeur maximale qui dépend de la quantité de mémoire installée sur votre ordinateur.
Quand un programme déclare une variable, le compilateur réserve un emplacement mémoire avec une adresse unique pour stocker cette variable. L'adresse est associée au nom de la variable. Le compilateur utilisera cette adresse pour accéder au contenu de l'emplacement mémoire, de façon transparente pour le programmeur.

Utilisation

La déclaration d'un pointeur contient le type des données pointées et un astérisque devant le nom du pointeur :

```
nomtype *nomptr ;
```

`nomtype` représente le type de la variable pointée. L'astérisque (*) devant le nom de la variable est *l'opérateur indirect,* il indique que `nomptr` est un pointeur vers une variable de type `nomtype` et non pas une variable de type `nomtype`.

```
char *ptr_car ;
```

Une déclaration peut contenir des pointeurs et des variables :

```
int valeur, *ptr_valeur ;
```

Initialisation

Un **pointeur doit être initialisé**, travailler avec un pointeur non initialisé peut se révéler désastreux. Un pointeur doit contenir l'adresse d'une variable. *L'opérateur d'adresse* (&) permet d'obtenir l'adresse d'une variable.

```
ptr_valeur =&valeur; /* ptr_valeur reçoit l'adresse de la variable valeur*/
```

Emploi

L'opérateur indirect (*) permet d'accéder au contenu d'une variable si on place cet opérateur devant le nom du pointeur vers cette variable.
Les instructions suivantes imprimeront la même valeur :

```
printf ("%d", valeur);      /* Acces direct */
printf ("%d", *ptr_valeur); /* Acces indirect */
```

Les 2 instructions suivantes initialisent la variable valeur avec la valeur 1 :

```
valeur = 1;
*ptr_valeur = 1; /* Si on a initialise ptr_valeur = &valeur  */
```

(Attention : si on exécute *ptr_valeur = 1 sans avoir initialisé ptr_valeur, on va écrire n'importe où dans la mémoire.)

`valeur` et `*ptr_valeur` représente le contenu de valeur.
`ptr_valeur` et `&valeur` représente l'adresse de valeur.

Pointeurs et types de variables

Les différents types de variables du langage C n'occupent pas tous la même taille mémoire. La taille mémoire de chaque type de variable est fixée par le compilateur et dépend de l'architecture de l'ordinateur. On peut connaître la taille mémoire de chaque type ou de chaque variable grâce à la fonction sizeof. Exemples : `sizeof (int)` donne la taille mémoire d'un entier et `sizeof(valeur)` donne la taille mémoire de la variable valeur.
Chaque octet de la mémoire a une adresse différente. Un pointeur sur une variable pointe sur le premier octet occupé par cette variable.

Exercice

6.1 Écrire un programme qui déclare une variable de type int et un pointeur sur une variable de type int.

- Initialiser la variable et imprimer son contenu par la méthode habituelle et en utilisant le pointeur.
- Afficher l'adresse de la variable à l'aide de son nom et en utilisant le pointeur.

Pointeurs et tableaux

Les pointeurs sont très utilisés pour travailler avec les tableaux.
Le nom d'un tableau est un pointeur vers le premier élément du tableau.

Déclaration d'un tableau et réservation de mémoire pour 10 entiers :

```
int data [10] ;
```

Le nom du tableau `data` est son adresse :
L'instruction suivante déclare un pointeur et réserve un emplacement mémoire pour une adresse soit 4 octets :

```
int *ptr_data ;
ptr_data = data ; /* Initialise ptr_data avec l'adresse de data */
```

Cette initialisation est équivalente à :

```
ptr_data = &data[0] ;
```

L'adresse d'un tableau est l'adresse de son premier élément.

Stockage des éléments d'un tableau

Les éléments d'un tableau sont stockés dans des emplacements mémoire séquentiels, le premier élément ayant l'adresse la plus basse. L'adresse de l'élément suivant dépend de la taille des valeurs stockées dans le tableau.

Dans un tableau de type `int`, chaque élément occupe 4 octets. Le pointeur sera incrémenté de 4 pour passer d'un élément à l'élément suivant.

```
int table[3], i;
for (i=0 ; i<3 ; i++) {
    printf ("Adresse de table[%d] : %d\n",i, &table[i]);
}
```

Donne le résultat suivant :

```
Adresse de table[0] : 6684140
Adresse de table[1] : 6684144
Adresse de table[2] : 6684148
```

Dans un tableau de type char, chaque élément occupe 1 octet. . Le pointeur sera incrémenté de 1 pour passer d'un élément à l'élément suivant.

```
char table[3];
int i;
for (i=0 ; i<3 ; i++) {
    printf ("Adresse de table[%d] : %d\n",i, &table[i]);
}
```

Donne le résultat suivant :

```
Adresse de table[0] : 6684148
Adresse de table[1] : 6684149
Adresse de table[2] : 6684150
```

Opérations sur les pointeurs :

Incrémenter un pointeur et décrémenter un pointeur:

On peut incrémenter un pointeur, si on ajoute la valeur 1 à un pointeur, le pointeur pointera sur l'élément suivant du tableau.

```
pointeur ++ ; /* Equivalent a pointeur += 1 ou pointeur = pointeur + 1 */
```

Si le pointeur pointe sur un tableau de type `int`, la valeur du pointeur sera augmentée de 4.
Si le pointeur pointe sur un tableau de type `char`, la valeur du pointeur sera augmentée de 1.

Le pointeur est augmenté de la taille du type des éléments adressés.

```
pointeur += 2; /* Equivalent a pointeur = pointeur + 2 */
```

Le pointeur est augmenté de 2 fois la taille des éléments pointés.
La décrémentation suit la même règle que l'incrémentation. Le pointeur est décrémenté de la taille des éléments pointés.

```
pointeur -= 4 ; /* Equivalent a pointeur = pointeur - 4 */
```

Le pointeur est décrémenté de 4 fois la taille des éléments pointés.

Autres opérations :
Différence : On peut faire la différence de 2 pointeurs pointant sur le même tableau et on obtiendra le nombre d'éléments qui sépare les 2 pointeurs.

Comparaison (==) : On peut comparer 2 pointeurs. Il y a égalité si les pointeurs pointent sur le même emplacement mémoire.

Exercice :
6.2 Écrire un programme qui lit des nombres entiers au clavier et qui les range dans un tableau. Le programme s'arrête sur la saisie de la valeur 0 ou lorsque le tableau est plein.
Écrire le programme en utilisant un pointeur sur le tableau (ne pas utiliser d'index pour accéder aux éléments du tableau).
Imprimer ensuite la liste des valeurs du tableau.

Pointeurs dans un tableau à plusieurs dimensions :
Soit un tableau à n dimensions, le nom du tableau utilisé avec n-1 index est un pointeur sur un tableau à une dimension.
Exemple :

```
int tab [2][3] = {{1, 2, 3}, {4, 5, 6}};
int *p;
p = tab [1]; /* tab[1] est un pointeur sur la 2ème ligne */
printf ("%d", *p);
```

Ce programme imprime la valeur 4.

Exercice :

6.3 Que va imprimer ce programme :

```
#include <stdio.h>
#define LG 10

void main () {
      char tableau1[LG]= {'0','1','2','3','4','5','6','7','8','9'};
      char tableau2[LG];
      char *pointeur1, *pointeur2;
      int i;

      pointeur1 = tableau1;
      pointeur2 = tableau2+LG;
      for (i = 0 ; i < LG ; i++) {
            pointeur2--;
            *pointeur2 = *pointeur1;
            pointeur1++;
      }
      for (i = 0 ; i < LG ; i++) {
            printf ("%c ", *pointeur2);
            pointeur2++;
      }
}
```

7. Chaînes de caractères

Définition
Un caractère (type char) peut être une lettre, un chiffre, une marque de ponctuation ou tout autre symbole. Un caractère est une valeur numérique stockée sur 1 octet (8 bits). Par exemple : la valeur 97 en décimal correspond à la lettre 'a' dans le code ASCII.
On utilise des tableaux de caractères pour stocker les chaînes de caractères.
Une chaîne de caractères est une séquence de caractères qui se termine par le caractère '\0'.
Bien qu'il soit représenté par 2 caractères (antislash et zéro), le caractère nul est interprété comme un seul caractère et sa valeur ASCII est 0.

Utilisation
```
char chaine[7] ; /* Permet de stocker une chaine de 6 caractères plus le caractère null */
```
Pour stocker la chaîne "Paris", il faut déclarer un tableau de 6 caractères.

Initialisation
Les tableaux de caractères peuvent être initialisés dans l'instruction de déclaration de cette façon :
```
char chaine[6] = {'P','a','r','i','s','\0'} ;
ou
char chaine[] = {'P','a','r','i','s','\0'} ;
ou
```
Avec utilisation d'une *chaîne littérale*, séquence de caractères entre guillemets :
```
char chaine[10] = "Soleil" ; /*Le caractère '\0' est mis en fin de chaine */
ou
char chaine[] = "Lune" ; /* Le caractère '\0' est mis en fin de chaine */
```

Comme pour les tableaux, quand on utilise le nom du tableau (`chaine`) sans indexation, on a un pointeur vers le premier caractère de la chaîne.

```
char *ptr_chaine = "Paris" ;
```
Dans ce cas, le compilateur réserve un emplacement pour un pointeur **et** un emplacement pour la chaîne de caractères "Paris" **et** initialise le pointeur par l'adresse du premier caractère de la chaîne.

Exercices :

7.1- Combien d'octets seront alloués pour stocker les variables suivantes :
1- char *ch1 = "Paris";
2- char ch2[] = "12345";
3- char ch3[20] = "Bonjour";
4- char ch4[20];
5- char ch5;

7.2- Soit l'instruction suivante :
char *ch = "Ceci est un exemple";
Quelles sont les valeurs des expressions suivantes :
1- ch[0]
2- *ch
3-ch[25]
4-*ch+5
5- *(ch+5)
6- ch[19]

Emploi :

Fonction puts
Imprime la chaîne de caractères et ajoute '\n'.

```
char message[] = "Paris" ;
puts (message) ; /* message est l'adresse de la chaine de caractères */
```

Fonction printf
Pour imprimer une chaîne de caractères avec la fonction printf, on utilise le format "%s". Cet argument reçoit un pointeur vers le début de la chaîne à imprimer. La fonction printf va afficher tous les caractères à partir de cette adresse, jusqu'à ce qu'elle rencontre la valeur 0.

```
char *message = "Paris" ;
printf ("%s", message) ; /*message est l'adresse de la chaine de caractères
*/
```

Fonction gets
Cette fonction lit une ligne de caractères entrée au clavier par l'utilisateur. Elle lit tout ce qui est tapée sur le clavier, jusqu'à ce que l'utilisateur enfonce la touche 'Entrée'. Le caractère de retour à la ligne est remplacé par le caractère '\0' et la chaîne est stockée à l'adresse qui a été passée en argument de la fonction.

```
char ligne[81];
gets (ligne) ;/*La chaine saisie ne doit pas avoir plus que 80 caractères*/
```

Fonction fgets
Cette fonction lit une ligne de caractères saisie au clavier comme gets mais en limitant le nombre de caractères saisis. Le caractère '\n' n'est pas remplacé et restera dans la chaîne de caractères. Le caractère '\0' sera rajouté derrière le dernier caractère.

```
char ligne[81];
fgets (ligne, 81, stdin) ;
```

Dans cet exemple, la fonction fgets permet de saisir au plus 80 caractères car elle garde toujours un caractère pour rajouter '\0'. Si la ligne saisie au clavier comporte plus que 80 caractères, fgets prendra les 80 premiers caractères saisis et ajoutera le caractère '\0'.

Fonction scanf
Le format "%s" permet de lire une chaîne de caractères qui sera stockée à l'adresse passée en argument. Le premier caractère de la chaîne sera le premier caractère non blanc qui sera lu au clavier et la chaîne s'arrêtera au premier caractère blanc rencontré ou au caractère 'Entrée'.

```
char ligne [81] ;
scanf (" %s ", ligne) ;
```

On peut limiter le nombre de caractères de la chaîne lue en précisant le nombre maximum de caractères attendus :

```
char codepostal [5] ;
scanf (" %4s ", codepostal) ;
```

Dans ce cas, les 4 premiers caractères avant un espace seront lus et le caractère '\0' sera ajouté. La fonction s'arrêtera au premier blanc rencontré s'il arrive avant d'avoir les 4 caractères.
Si on tape 11124, codepostal prendra la valeur 1112.
Si on tape 111 24, codepostal prendra la valeur 111.
La fonction scanf ne permet pas de lire une chaîne de caractères vide.

Fonction sscanf
Cette fonction s'utilise de la même manière que scanf mais elle utilise une chaîne de caractère à la place de la saisie au clavier.

```
char chaine [] = "123 456" ;
int a, b ;
sscanf (chaine, "%d %d", &a, &b) ;
/* La variable a recevra la valeur 123 et la variable b aura la valeur
456*/
```

Fonction strlen
Pour utiliser cette fonction, il faut :

```
#include <string.h>
```

La fonction donne la longueur de la chaîne de caractères (non compris le caractère '\0').

```
char chaine [] = "Paris";
printf ("Longueur de la chaine : %d\n", strlen(chaine)) ;
```

Donne le résultat suivant :

```
Longueur de la chaine : 5
```

7.3- Écrire un programme

- qui lit 2 chaînes de caractères (avec une seule fonction scanf)
- qui les imprime : chaîne1 : toto, chaîne2 : tata
- et qui dit si elles sont identiques.

Fonctions utilitaires sur les chaînes de caractères

Les chaînes de caractères sont très utilisées dans les applications. Exemples : Recherche d'un nom dans une liste de personnes, insertion du prénom devant le nom d'une personne.
Les chaînes de caractères nécessitent des opérations de comparaisons, de copies, de concaténations qui ne sont pas incluses dans le langage C. Ces opérations sont réalisées par des fonctions de librairie qui sont définies dans le fichier d'entête <string.h>.

Concaténation

`char *strcat (s1, s2)` : concatène la chaîne s2 à la suite de la chaîne s1 et retourne s1.

`char *strncat (s1, s2, n)` : concatène au plus n caractères de la chaîne s2 à la suite de la chaîne s1, termine s1 par '\0' et retourne s1.

Attention dans les 2 cas, le tableau de caractères s1 doit avoir la taille suffisante.

```
char ch1[50] = "Bonjour ";
char ch2[50] = "Monsieur";
strcat (ch1, ch2) ;
printf (" Resultat %s\n", ch1);
```

Donne le résultat suivant :

```
Bonjour Monsieur
```

Et :

```
strncat (ch1, ch2, 5) ;
printf (" Resultat %s\n", ch1);
```

Donne le résultat suivant :

```
Bonjour Monsi
```

Copie

`char *strcpy (destin, source)` : copie la chaîne `source` (y compris '\0') dans la chaîne `destin` et retourne `destin`.

`char *strncpy (destin, source, n)` : copie au plus n caractères de la chaîne `source` dans la chaîne `destin`, complète par des '\0' si `source` comporte moins de n caractères et retourne `destin`. Attention, si `source` comporte plus de n caractères, le caractère '\0' ne sera pas recopié.
Attention dans les 2 cas, le tableau de caractères `destin` doit avoir la taille suffisante.

```
char ch1[] = "xxxxxxxxxxxxxxxxxxx";
char ch2[50];
printf (" Donner un mot : ");
gets (ch2) ;
strncy (ch1, ch2, 7) ;
printf ("Resultat : %s",ch1);
```

Donne les résultats suivants :

```
Donner un mot : Bon
Resultat : Bon
```

Et :

```
Donner un mot : Bonjour
Resultat : Bonjourxxxxxxxxxxxxx
```

Comparaison

`int strcmp (s1,s2)` : compare les 2 chaînes de caractères `s1` et `s2`.

`int strncmp (s1,s2,n)` : travaille comme strcmp mais limite la comparaison à n caractères.

Le résultat est une valeur entière définie ainsi :

```
>0 si s1 > s2
==0 si s1 == s2
<0 si s1 < s2
```

Exemple :

```
char chaine1 [20], chaine2[20] ;
printf ("Entrer 2 mots");
scanf ("%19s %19s", chaine1, chaine2);      // Saisie de 2 mots au clavier
if (strcmp (chaine1, chaine2) == 0) {       // Comparaison des 2 mots
    printf ("Les 2 mots sont identiques");
} else {
    printf ("Les 2 mots sont différents");
}
```

Recherche dans une chaîne

`char *strchr (chaine, car)` : recherche dans `chaine` de la première position où apparaît le caractère `car`.

`char *strrchr (chaine, car)` : même traitement que `strchr` mais en commençant par la fin.

`char *strstr (chaine1, chaine2)` : recherche dans `chaine1` la première position où apparaît la première occurrence complète de `chaine2`.

Le résultat est **l'adresse** de l'information cherchée et le pointeur NULL dans le cas contraire.

Exemple :

```
char texte [100];
char *ptr; // pointeur utilisé pour le résultat de la fonction strchr
int nb ;

nb = 0 ;
gets (texte);
ptr = strchr (ligne, 'i');
/* ptr aura l'adresse du premier 'i' trouvé dans texte */
/* ou voudra NULL si il n'y a pas de i dans texte */
while (ptr != NULL) {          // Tant qu'un caractère i a été trouvé
      nb++ ;
      ptr = strchr (ptr + 1, 'i') ;
      /* Recherche du 2ème i dans le texte – la recherche démarre */
      /* après le 1er 'i' */
}
```

Exercices :

7.4 Écrire un programme déterminant le nombre de lettre e (minuscule) contenues dans une ligne de texte saisie au clavier (utiliser la fonction strchr) et remplacer tous les e par le signe '-'.

7.5 Écrire un programme qui supprime toutes les lettres 'e' (minuscule) d'un texte d'une ligne saisie au clavier. Le texte modifié écrasera la ligne de texte saisie. (Utiliser la fonction strchr pour chercher le caractère 'e' et utiliser strcpy pour recopier la chaîne de caractères.)

7.6 Remplacer toutes les occurrences d'un mot par un autre mot dans une ligne de texte saisi au clavier :

```
Mot a remplacer : renard
Nouveau mot : loup
Entrer une ligne de texte :
=> Le renard et le corbeau. Le renard a faim et le corbeau a un fromage.
Resultat : Le loup et le corbeau. Le loup a faim et le corbeau a un
   fromage.
```

Vous devez lire la ligne de texte dans un tableau de caractères et construire la nouvelle chaîne de caractères dans un deuxième tableau. Imprimer le résultat.

7.7 Donner le résultat exact de ce programme :

```
#include <stdio.h>

#include <string.h>

void main ()
   char message1 [50];
   char message2 [50];
   char chaine1 [] = "Nous irons au bois";
   char chaine2 [] = "Lise et Paul";

   strcpy (message1, "Zero Un Deux");
   puts (message1);

   strcpy (message1, "Quatre Cinq");
   puts (message1);

   strcat (message1, chaine1);
   puts (message1);

   strcpy (message2, chaine1);
   puts (message2);

   strcat (message2, " Paul");
   puts (message2);

   strncpy (message2, chaine2, 4);
   puts (message2);

   printf ("%d %d\n", sizeof (chaine2), strlen (chaine2));
}
```

8. Fonctions de librairies

Tests sur les caractères

Ces fonctions retournent une valeur non nulle (vrai) si l'argument c remplit la condition indiquée et zéro dans le cas contraire.
Les fonctions suivantes nécessitent une directive include avec le fichier ctype.h.

Fonction	*Condition*
`isalpha(c)`	Caractère alphabétique
`isdigit(c)`	Chiffre décimal
`isalnum(c)`	Caractère alphabétique ou chiffre décimal
`islower(c)`	Lettre minuscule
`isupper(c)`	Lettre majuscule
`ispunct(c)`	Caractère imprimable différent de l'espace, des lettres et des chiffres
`isspace(c)`	Espace, saut de page, fin de ligne, retour chariot, tabulation

Conversion de caractères

Fonction	*Type du résultat*	*Action*
`toupper(c)`	char	Retourne c en majuscule ou c si c est majuscule
`tolower(c)`	char	Retourne c en minuscule ou c si c est minuscule
`atoi(chaine)`	int	Conversion d'une chaîne en entier (int)
`atof(chaine)`	double	Conversion d'une chaîne en double

Toupper et tolower nécessitent le fichier <ctype.h> et atoi et atof le fichier <stdlib.h>

Exemples

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
void main () {
      char chaine[100];
      unsigned i;
      int nbLettre, nbChiffre, nbMaj, nbMin;

      nbLettre = 0;  /* Nombre de lettres */
      nbChiffre = 0; /* Nombre de chiffres */
      nbMaj = 0;     /* Nombre de majuscules */
      nbMin = 0;     /* Nombre de minuscules */
      printf ("Entrer une ligne de texte :\n");
      fgets (chaine, 100, stdin);
      for (i=0 ; i<strlen(chaine) ; i++) {
            if (isalpha (chaine[i]) ){
                  nbLettre++;
                  if (islower(chaine[i]))
                        nbMin++;
                  else
                        nbMaj++;
            } else if (isdigit(chaine[i]))
                  nbChiffre++;
      }
      printf ("Nombre de lettres    : %d\n", nbLettre);
      printf ("Nombre de majuscules : %d\n", nbMaj);
      printf ("Nombre de minuscules : %d\n", nbMin);
      printf ("Nombre de chiffres   : %d\n", nbChiffre);
}
```

Exercices :

8.1 Écrire un programme qui lit une ligne de texte au clavier et qui convertit :

- les majuscules en minuscules
- les minuscules en majuscules
- les espaces, tabulations et ‘\n’ en caractère ‘-‘.

Utiliser 2 tableaux de caractères : un tableau pour lire la ligne de texte et un tableau pour construire la chaîne de caractères après les modifications (que vous devrez imprimer).

9. Tableau de pointeurs

Déclaration

```
type *tab [n] ;
```
type est le type des éléments pointés par les pointeurs, éléments du tableau .

Utilisation

```
char *jour [] =
{"lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"};
```
Cette déclaration définit un tableau de 7 pointeurs initialisés respectivement avec les 7 adresses des chaînes littérales.
```
char *liste_noms[20] ;
```
Cette instruction réserve un tableau de 20 pointeurs sur des chaînes de caractères sans initialisation.
```
int *tab_pointeurs[25] ;
```
Cette instruction réserve un tableau de 25 pointeurs sur des entiers.

Pointeur sur un tableau de pointeurs

Un tableau de pointeurs peut être accédé par un indice comme les autres tableaux (ex jour[i]) et aussi en utilisant un pointeur :
```
char **ptr_tab = jour;
```
`ptr_tab` pointe sur le premier élément du tableau jour qui est un tableau de pointeurs. Il faut 2 caractères '*' consécutifs pour déclarer un pointeur sur des pointeurs. Le pointeur `ptr_tab` ne pointe pas sur des caractères mais sur des pointeurs de caractères.
`*ptr_tab` est la valeur du premier élément du tableau de pointeur (pointeur sur la chaîne de caractères "lundi" dans l'exemple).

`**ptr_tab` est le 1er caractère pointé par le premier élément du tableau (est égal à 'l' premier caractère de lundi).

Exercices :

9.1 Lire le nom d'un jour de la semaine (lundi à dimanche) et donner le numéro du jour dans la semaine :
```
      Jour : mercredi
      mercredi est le 3ème jour de la semaine
```

9.2 Quel est le résultat de ce programme :
```
#include <stdio.h>
void main () {
      char *chaines [] = {"NBNCND","MRMPMQ","TATBTC","AVAWAX","ZOZPZQ"};
      int i, j, max;
      j = 1;
      max = sizeof(chaines) / sizeof(char *);
      for (i = 0; i < max; i ++) {
            printf ("%c", chaines [i] [j]);
      }
      printf ("\n");
}
```

9.3 Écrire un programme qui saisit un nombre entre 1 et 7 au clavier et qui affiche le nom du jour de la semaine correspondant (lundi pour 1, etc…)

9.4 Écrire un programme qui lit une ligne de chiffres écrit en lettres et les traduit en chiffres arabes :

```
Tapez une série de chiffres en lettres : un trois quatre sept
Resultat : 1 3 4 7
```

Le programme se termine sur une ligne vide.

10 Allocation dynamique de mémoire

L'allocation dynamique de mémoire est utilisée lorsqu'on ne connaît pas à la compilation la taille mémoire dont aura besoin le programme pour s'exécuter.

Utilisation

```
#define LGMAX 30
char *pointeur;
pointeur = (char *) malloc (LGMAX);
```

La fonction malloc donne l'adresse d'un bloc mémoire comportant le nombre d'octets passé en paramètre. La fonction malloc nécessite le fichier <stdlib.h>. On doit faire un 'cast' sur le résultat de malloc pour lui donner le bon type. On peut ensuite utiliser cette zone mémoire :

```
scanf ("%s", pointeur) ;
printf ("Voici la chaine lue : %s\n ",pointeur) ;
```

Exemple de réservation de mémoire pour un tableau d'entier :

```
#define NBENTIER 30
int *pointeur;
pointeur = (int *) malloc (NBENTIER*sizeof (int));
/* Nbre d'éléments * taille d'un entier */
for (i=0 ; i< NBENTIER ; i++) { pointeur[i]=33 };
```

Libérer la mémoire allouée dynamiquement

Lorsque le programme n'a plus besoin d'une zone mémoire allouée dynamiquement, il doit la libérer pour ne pas bloquer la mémoire inutilement.

```
char *pointeur;
pointeur = (char *) malloc (LGMAX);
free (pointeur);
```

La fonction `free` peut être appelée seulement avec des valeurs qui ont été donnée par la fonction `malloc`. Une zone mémoire libérée ne peut plus être utilisée.

Exercices :

10.1 Écrire un programme qui saisit une commande :
- -1 pour entrer un nom
- et 0 pour sortir.

Si la commande est 1 : lire un nom au clavier et le stocker dans une liste.
Si le nom existe déjà dans la liste, on imprime un message d'erreur.
A la sortie, on imprime la liste et on libère la mémoire allouée.

```
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : Paris
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : Oslo
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : New York
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : Vienne
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : Berne
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : New York
Le nom est deja dans la liste
Commande (1 pour entrer un nom, 0 pour sortir) : 1
Entrer un nom : Las Vegas
Commande (1 pour entrer un nom, 0 pour sortir) : 0
Nom numero 0 : Paris
Nom numero 1 : Oslo
Nom numero 2 : New York
Nom numero 3 : Vienne
Nom numero 4 : Berne
Nom numero 5 : Las Vegas
```

11 Les fonctions

Définition

Une fonction est un bloc d'instructions, référencé par un nom, qui réalise une tâche et qui peut renvoyer une valeur au programme qui l'a appelée.
Le nom d'une fonction est un identificateur, on peut exécuter le code de la fonction en utilisant son nom. Une fonction peut être appelée par une autre fonction.
Une fonction peut recevoir des informations de la part du programme appelant et elle peut renvoyer un résultat au programme appelant.

Exemple

```
/* Exemple d'une fonction simple */
#include <stdio.h>
/* Prototype ou declaration de la fonction */
int cube (int x) ;

void main () {
      int val;
      int resultat;
      printf ("Entrer une valeur : ");
      scanf ("%d", &val);
      /* Appel de la fonction cube avec un argument */
      resultat = cube (val);
      printf ("Le cube de %d est %d\n", val, resultat);
}

/* Definition de la fonction cube */
/* La fonction a un argument x et rend un résultat de type int */
int cube (int x) {
      /* Declaration d'une variable locale */
      int cube_x;
      cube_x = x * x * x;
      /* Renvoi d'une valeur */
      return (cube_x);
}
```

Fonctionnement

Les instructions d'une fonction ne sont exécutées dans un programme que lorsque la fonction est appelée par une autre partie de ce programme. Quand la fonction est appelée, le programme lui transmet des informations sous la forme d'arguments. Un argument est une donnée dont la fonction a besoin pour exécuter sa tâche (x dans l'exemple). La fonction s'exécute et le programme reprend à partir de l'instruction qui contient l'appel, en récupérant la valeur de retour (cubed_x dans l'exemple) si il y en a une.
Les fonctions peuvent être appelées autant de fois que nécessaire.

Le prototype de la fonction fournit au compilateur la description d'une fonction qui est définie plus loin dans le programme. Cette description comprend

- le nom de la fonction,
- le type de la valeur de retour,
- les types des arguments (et en option leurs noms)

```
type_retour nom_fonction (type_arg1 nom-1,...., type_argn nom-n) ;
```

Exemple de prototype :

```
int surface_rectangle(int largeur, int longueur) ;
```

La définition de la fonction est le code qui sera exécuté à l'appel de la fonction. La première ligne est ***l'en-tête de la fonction*** et doit être cohérente avec le prototype de la fonction. Cet en-tête contiendra le nom des arguments (qui est en option dans le prototype), il est suivi par le bloc d'instructions de la fonction entre accolades. Si le type de la valeur de retour n'est pas `void`, il faut une instruction `return` qui renvoie une valeur du type défini.

```
type_retour nom_fonction (type_arg1 nom-1,...., type_argn nom-n) {
      /* Instructions */
      return valeur ;
}
```

L'exécution de la fonction commence à la première instruction du corps de la fonction et se termine à la première instruction return rencontrée ou à l'accolade de fin.

Les variables locales sont déclarées après l'en-tête de la fonction. Elles existent seulement pendant l'exécution de la fonction et disparaissent lorsque la fonction est terminée. La fonction n'a pas accès aux variables déclarées dans la fonction `main` et dans les autres fonctions. Pour donner accès à la valeur d'une variable, il faut la passer en paramètre. Le nom d'une variable locale peut être le même que le nom d'une variable déclarée dans la fonction `main`, la fonction ne verra que la variable déclarée localement.

Les arguments sont transmis à une fonction au moyen de la liste entre parenthèses qui suit le nom de la fonction. Le nombre et le type de ces arguments doivent correspondre aux indications du prototype et de l'entête. Une fonction peut avoir une liste d'argument vide.

Le résultat de la fonction est transmis au programme appelant par l'instruction `return`. Il est recommandé d'utiliser une seule instruction `return` dans le corps de la fonction. Si le type du résultat de la fonction est `void`, il n'y a pas d'instruction `return`.

Exemples :

```
void ecritBonjour () {
 puts ("Bonjour"); /*Cette fonction n'a pas d'argument et pas de resultat*/
}
int fonct1 (int a, int b) {
  return (a * b) ;  /* Le resultat est la multiplication des 2 arguments */
}
```

Placement des fonctions

Les prototypes des fonctions sont définis avant la fonction `main`, et les définitions des fonctions viennent après la fonction main :

```
/* Debut du code */
...
/* prototypes des functions */
typ1 fonct1(type_arg1 arg1, type_arg2 arg2,…);
...
void main () {
      ...
      ...
}
type1 fonct1(type_arg1 arg1, type_arg2 arg2,…) {
...
}
type2 fonct2() {
}
```

Exercices :

11.1 Écrire une fonction qui reçoit 2 nombres de type float et qui rend le plus grand. Écrire le programme principal qui saisit 2 nombres et qui appelle la fonction pour imprimer la plus grande valeur.

11.2 Écrire la fonction : `int factorielle (int val) ;`
qui calcule factorielle (val). Écrire la fonction main qui appelle cette fonction après avoir saisi une valeur entière et imprimer le résultat.

11.3 Écrire la fonction : `int puissance (int val, int n) ;`
qui calcule val à la puissance n. Écrire la fonction main qui appelle cette fonction après avoir saisi les 2 valeurs entières val et n et imprimer le résultat de la fonction (dans la fonction main).

11.4 Écrire la fonction : `int sommeNbEntier (int n) ;`
qui calcule la somme des n premiers nombres entiers. Écrire la fonction main qui appelle cette fonction après avoir saisi la valeur entière n et imprimer le résultat de la fonction (dans la fonction main).

11.5 Écrire une fonction qui saisit au clavier une valeur entière comprise entre une valeur minimum et une valeur maximum. La fonction aura 2 arguments min et max. Elle donne un message d'erreur si la valeur n'est pas bonne ou si la saisie est incorrecte. La fonction redemandera une valeur tant que la valeur n'est pas comprise entre les 2 bornes.
- Écrire la fonction `premier` qui a comme argument une valeur entière positive et qui rend la valeur VRAI (1) si la valeur est un nombre premier et FAUX (0) dans le cas contraire.
- Écrire la fonction main qui appelle la première fonction pour saisir un nombre entre 1 et 100 et qui appelle la deuxième fonction pour afficher tous les nombres premiers entre 1 et la valeur saisie.

Passage des paramètres

Par défaut, les paramètres de type de base (char, int, float) sont passés par valeur : L'expression passée en paramètre est évaluée et le résultat de l'évaluation est recopié dans le paramètre. Les paramètres sont utilisés comme des variables locales. Ils sont toujours initialisés à l'entrée de la fonction. Une fonction ne peut pas modifier des variables qui ne sont pas visibles par le corps de la fonction.
Lorsqu'on passe un tableau en paramètre d'une fonction, on passe son adresse. On ne peut pas passer un tableau par valeur.

Type de donnée	**Mode de transmission**
Type de base : char, float, int, long, double	Passage par valeur
Structures	Passage par valeur
Tableau	Passage par adresse

Pointeurs et passage par adresse

Lorsqu'une fonction a besoin de modifier une variable qui est visible par la fonction appelante mais pas par la fonction appelée, on utilise la méthode de passage par adresse. Les arguments passés par adresse doivent être déclarés comme des pointeurs sur des variables et dans le corps de la fonction, il faut utiliser l'opérateur d'indirection (*) pour accéder à la variable. L'opérateur '&' permet de passer les adresses des variables. Les adresses des variables sont copiées dans la pile et la fonction peut les utiliser pour accéder en lecture et en écriture.

Tableaux en paramètre

Pour passer un tableau en paramètre, on indique le nombre de dimensions. Pour chaque dimension, on doit préciser la taille entre crochet. La taille de la dimension est facultative pour la 1ère dimension.

```
int somme (int table[], int n); // tableau à une dimension en paramètre

int tableau [100];
int s;
s = somme (tableau, 100);

int somme (int table[][10], int nbl, int nbc) ;
// paramètre : tableau à 2 dimensions

int tableau [100][10];
int s ;
s = somme (tableau, 100, 10) ;
```

Résultat d'une fonction

Il peut être de n'importe quel type de base (char, float, int, long, double) ou une structure ou un pointeur.

Exemple d'une fonction avec un tableau en paramètre :

```
#include <stdio.h>
/* Fonction qui calcule la somme des n éléments du tableau */
int sommeTab (int *tab, int n);

void main () {
      int tab [5] ={1,2,3,4,5};
      printf ("La somme est : %d\n", sommeTab (tab, 5 ));
}
int sommeTab (int *tab, int n) { /*tab est l'adresse du premier element */
      int i;
      int res = 0;

      for (i=0 ; i < n; i++) {
            res += tab [i];
      }
      return res;
}
```

Exercices :

11.6- Écrire une fonction 'min' qui reçoit en paramètre : un tableau d'entiers et le nombre d'éléments de ce tableau. Le résultat de cette fonction est la plus petite valeur de ce tableau.
Écrire le programme principal qui saisit des valeurs entières au clavier et qui initialise un tableau d'entiers. Appeler la fonction précédente pour rechercher la valeur minimale et imprimer le résultat de la fonction.

11.7 Écrire la fonction : `void echange (int *pta, int *ptb)` ; qui échange la valeur des 2 variables pointées par pta et ptb. Écrire le programme principale qui initialise 2 variables de type int et qui appelle la fonction echange pour échanger leur valeur. Imprimer les valeurs des variables avant l'échange et après l'échange.

11.8 Écrire la fonction : `void tri (int *pta, int *ptb, int *ptc)` ; qui échange les valeurs des 3 variables dont les adresses sont pta, ptb et ptc pour obtenir les 3 valeurs dans l'ordre croissant (*pta <= *ptb <= *ptc). Vous devez utiliser la fonction echange écrite dans l'exercice précédent pour faire les échanges de valeurs 2 à 2. Écrire le programme principale qui initialise 3 variables de type int et qui appelle la fonction tri. Imprimer les valeurs des variables avant l'appel à la fonction tri et après l'appel à la fonction tri.

11.9 Écrire les fonctions suivantes :

int `sommeLigne (int tab [NBLIG][NBCOL], int i) ;` qui calcule la somme des éléments de la ligne numéro i dans le tableau à 2 dimensions tab qui comporte NBCOL colonnes (constante définie par la directive define).

`int sommeColonne (int tab [NBLIG][NBCOL], int j) ;` qui calcule la somme des éléments de la colonne numéro j dans le tableau à 2 dimensions tab qui comporte NBLIG colonnes (constante définie par la directive define).

Écrire la fonction principale qui déclare un tableau à 2 dimensions et qui l'initialise à l'aide de valeurs entières saisies au clavier. Afficher ligne par ligne les valeurs du tableau et la somme de chaque ligne suivies par la somme de chaque colonne.

Exemple :

```
Taille du tableau 24 octets
Entrer la valeur no :  1 (table[0][0])4
Entrer la valeur no :  2 (table[0][1])5
Entrer la valeur no :  3 (table[0][2])6
Entrer la valeur no :  4 (table[1][0])1
Entrer la valeur no :  5 (table[1][1])2
Entrer la valeur no :  6 (table[1][2])4

|     4 |     5 |     6 |    15
|-------|-------|-------|-------
|     1 |     2 |     4 |     7
|-------|-------|-------|-------
|     5 |     7 |    10
```

11.10 Écrire la fonction suivante :

```
void addMatrice (int mat1[NBL][NBC],
                 int mat2[NBL][NBC],
                 int mat3[NBL][NBC]) ;
```

où NBL et NBC sont définies par des directives define.
La fonction additionnera les 2 matrices mat1 et mat2 et le résultat sera affecté à mat3.
Écrire le programme principal, qui initialise 2 matrices et qui appelle la fonction addMatrice pour calculer leur somme et afficher le résultat obtenu.

La portée des variables :

La portée d'une variable est la zone du programme dans laquelle la variable est visible ou accessible.

Le « temps de vie » d'une variable, c'est à dire le temps pendant lequel la donnée est conservée en mémoire dépend de la portée de la variable.

Variable globale : C'est une variable définie en dehors d'une fonction et en dehors de la fonction main. Elle est visible par toutes les fonctions qui sont définies après la déclaration de cette variable. Elle existe pendant toute la durée du programme.

Variable locale : C'est une variable définie dans une fonction ou dans la fonction main : elle est visible dans tout le corps de la fonction qui suit la déclaration de la variable. Elle existe au début de l'exécution de la fonction et elle disparaît à la fin de l'exécution de la fonction. Ce type de variable est aussi appelé variable automatique. Elle est allouée au début de la fonction dans laquelle elle est déclarée. Une variable peut être définie à l'intérieur d'un bloc dans une fonction. Dans ce cas, sa portée est limitée à ce bloc et sa durée de vie est limitée à l'exécution du bloc d'instructions. Les variables locales de la fonction main subsistent pendant toute la durée du programme.

Variable locale statique : Une variable peut être définie comme statique en utilisant le mot clé `static` devant la déclaration de la variable. Dans ce cas, la variable est visible dans le corps de la fonction comme une variable locale mais elle existe pendant toute la durée d'exécution du programme. Ce type de variable est utilisé pour garder des informations entre 2 appels de la même fonction.

Variable globale externe : Une variable sera déclarée comme `extern` pour pouvoir être exportée dans un autre fichier. On utilise ce type de variable dans la compilation séparée.

Nom des variables : 2 variables de même niveau de portée ne peuvent pas avoir le même nom. Des variables de niveau de portée différent peuvent avoir le même nom et c'est la variable de niveau inférieur qui sera visible : une variable locale cachera une variable globale en portant le même nom.

Exemple :

```
int val = 1;
#include <stdio.h>

/* Prototype des fonctions */
void fonction1 ();
void fonction2 ();

/* Cette variable peut être accédée par tout le reste du programme */
int varGlobale;

void main () {
      /* Apres cette déclaration val désignera cette nouvelle */
      /* variable et non plus la variable globale du même nom */
      int val = 2;

      varGlobale = 1;
      printf ("VarGlobale %d\n", varGlobale);
      printf ("Etape 1 val = %d\n", val);
      fonction1();
      fonction2();

}
void fonction1 () {
      varGlobale = 2;
      printf ("VarGlobale %d\n", varGlobale);
      /* Ici c'est la variable globale qui est visible */
      printf ("Etape 2 val = %d\n", val);
}
void fonction2 () {
      int val = 3;
      int i;

      printf ("Etape 3 val = %d\n", val);

      for (i=0; i<2 ; i++) {
            /* Exemple de variable locale à un bloc */
            /* A NE PAS FAIRE sauf exception */
            int val = 4; /* Cette variable disparaît chaque fin de boucle*/
            printf ("Etape 4 val = %d\n", val);
            val++;
            printf ("Etape 5 val = %d\n", val);
      }

}
```

Ce programme donne le résultat suivant :

```
VarGlobale 1
Etape 1 val = 2
VarGlobale 2
Etape 2 val = 1
Etape 3 val = 3
Etape 4 val = 4
Etape 5 val = 5
Etape 4 val = 4
Etape 5 val = 5
```

12. Les structures

Définition
Une structure contient une ou plusieurs variables groupées sous un même nom pour être traitées comme une seule entité. Chaque variable d'une structure est appelée ***membre*** de cette structure. Un membre de structure peut être de n'importe quel type (int, char, double etc..), ou un tableau ou une autre structure.

Utilisation
On a souvent besoin de manipuler un ensemble d'information sous une entité. Voici quelques exemples :

1- Données concernant l'identité d'une personne (nom, prénom, âge, adresse, etc)
2- Date : jour, mois, année
3- Coordonnées d'un point : x, y

La tâche du programmeur est simplifiée par l'utilisation d'une structure.

Déclaration des structures

```
struct nom_structure {
      liste des membres de la structure
} variable1, variable2,... ;
```

`struct` identifie le début de la structure .
`nom_structure` est le nom qui identifie la structure. Ce n'est pas le nom d'une variable.
`variable1, variable2` sont des variables de type `struct nom_structure`.

Exemples
Définition d'une structure sans déclaration de variable :

```
struct coord {
      int x ;
      int y ;
};
```

On pourra déclarer des variables de type `struct coord` avec la déclaration suivante :

```
struct coord point1, point2 ;
```

On peut aussi déclarer des variables directement lors de la définition de la structure :

```
struct coord {
      int x ;
      int y ;
} point1, point2;
struct heure {
      int heures ;
      int minutes ;
      int secondes ;
} heure_naissance = {14, 23, 5} ; // Initialisation statique
```

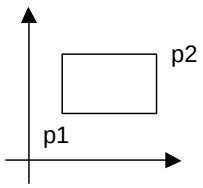
Accès aux membres d'une structure
Chaque membre d'une structure peut être utilisé comme une variable isolée du même type. On utilise l'opérateur '.' entre la variable de type structure et le nom du membre pour accéder au membre.

Exemple :
```
point1.x = 10 ;
point1.y = 50 ;
```
L'instruction suivante permet d'afficher les coordonnées du point :
```
printf ("Coordonnees du point %d %d\n", point1.x, point2.y);
```

Passage de structures comme argument de fonction
Une structure est transmise par valeur lorsqu'elle est passée en argument d'une fonction. Dans le corps de la fonction, on peut accéder aux membres de la structure.

Exemple :
```
struct coord {
      int x ;
      int y ;
} ;
void printPoint (struct coord point) {
      printf ("x : %d, y : %d\n", point.x, point.y) ;
}
```
Une structure est transmise par valeur lorsqu'elle est passée en argument d'une fonction. Dans le corps de la fonction, on peut accéder aux membres de la structure.

Exercice :
12.1 Soit la structure suivante :
```
struct coord {
      int x;
      int y;
} ;
```
- Intialiser 2 points statiquement définis avec ce type de structure.
- Écrire la fonction :
```
int surface (struct coord p1, struct coord p2) ;
```
et qui calcule la surface du rectangle définit ainsi :



Imprimer le résultat

Les structures plus complexes

Les membres d'une structure peuvent êtres d'autres structures, des tableaux ou des pointeurs.

Exemple de structures imbriquées :

```
struct rectangle {
      struct coord hautgauche ;
      struct coord basdroite ;
} boite;
```

Pour accéder aux membres de deux structures imbriquées, il faut utiliser deux fois l'opérateur '.' :

```
boite.hautgauche.x = 0 ;
boite.hautgauche.y = 10 ;
boite.basdroite.x = 100 ;
boite.basdroite.y = 200 ;
```

Exemple de définitions de structure dans une structure :

```
struct typeDate {
      struct typeHeure {
            int heures ;
            int minutes ;
      } heure ;
      struct typeJour {
            int noJour ;
            int mois ;
      } jour ;
} date ;
date.heure.heures = 8 ;
date.heure.minutes = 30 ;
date.jour.noJour = 23 ;
date.jour.mois = 6 ;
```

Exemple de structure composée de tableaux :

```
struct data {
      int x[3] ;
      char y[7] ;
} var1 = {1,2,3, "PARTEZ"), var2;
```

On peut ensuite utiliser var1 comme une seule entité :

```
var2 = var1 ;
strcpy (var2.y, "STOP");
var2.y[0] a la valeur 'S'.
```

Structure composée de 3 tableaux :

```
struct date {
      char jour[3];
      char mois[3];
      char annee[5];
} date_jour = {"14","09","1515"} ;
```

Initialisation statique des structures

Comme tout autre type de variable C, les structures peuvent être initialisées quand elles sont déclarées. La déclaration de la variable est suivie d'un signe égal puis, entre accolades, d'une liste de valeurs d'initialisation séparées par des virgules :

```
struct client {
      char societe[20] ;
      char contact[20] ;
};
struct vente {
      struct client acheteur;
      char article[20];
      float montant;
} mesventes = {{"Office world", "Gérard Lambert"},
                "papier imprimante", 100.00};
```

Les valeurs d'initialisation doivent apparaître dans l'ordre. Elles seront stockées dans les membres en utilisant l'ordre dans lequel ces membres sont listés dans la définition de la structure.

Test : Après cette initialisation, quelles sont les valeurs de :

```
mesventes.article[0]
mesventes.acheteur.contact[1]
mesventes.acheteur.societe
mesventes.montant
```

Opérations sur les structures

Une variable de type structure peut être initialisée par la valeur d'une variable du même type de structure. Contrairement aux tableaux en C, on peut copier une structure dans une autre structure en utilisant l'opérateur d'affectation.

```
struct vente tesventes;
tesventes = mesventes ;
```

Opérateur	Description
=	Affectation d'une structure à une structure du même type
.	Accès à un membre de la structure
&	Adresse d'une structure

Le jeu d'opérations possibles sur les membres d'une structure dépend du type des membres. Sur les membres de type int, toutes les opérations arithmétiques sont applicables. Pour les membres de type chaînes de caractères, il faut utiliser les fonctions de traitement de chaînes.

Exemples :

```
tesventes.montant *= 2; /* Le champ montant est multiplie par 2 */
strcpy (tesventes.acheteur.contact, mesventes.acheteur.contact);
```

Exercice :

12.2 Écrire la fonction : `void printVente (struct vente v)` qui imprime tous les champs de la structure vente et l'appeler pour les 2 variables `tesventes` et `mesventes` qui auront été initialisées statiquement comme dans les exemples précédents.

Pointeur sur une structure

Un programme C peut déclarer et utiliser des pointeurs vers des structures exactement comme il peut déclarer des pointeurs vers tout autre type de donnée. L'opérateur & est utilisé pour obtenir l'adresse d'une structure.

```
struct typePoint {
      int x;
      int y;
} point;
struct typePoint *ptrPoint;

ptrPoint = &point;
```

Pour accéder aux membres de la structure, on utilise l'opérateur '.' de cette façon :

```
(*ptrPoint).x = 10;
(*ptrPoint).y = 20;
```

Les parenthèses sont obligatoires car l'opérateur '.' est prioritaire sur l'opérateur '*'. Cette notation n'étant pas pratique, le langage C offre l'opérateur '->' permettant d'accéder au champ d'une structure à partir d'un pointeur.

Les 2 instructions suivantes sont équivalentes aux 2 instructions précédentes :

```
ptrPoint->x = 10;
ptrPoint->y = 10;
```

Pointeurs membres d'une structure

Les pointeurs peuvent aussi être membres d'une structure. Un pointeur membre d'une structure peut pointer sur une autre structure.

Exemple :

```
struct typeDate {
      int jour ;
      int mois ;
      int annee ;
} date1={14, 7, 1789};
struct typeAnniversaire {
      char *nom ;
      struct typeDate *date ;
} anniversaire ;
anniversaire.nom = "Jean Dupond" ;
anniversaire.date = &date1 ;
```

Le membre `nom` est initialisé avec l'adresse d'une chaîne littérale. Le membre date est initialisé avec l'adresse de la structure date1.

Les membres de la structure seront accédés de la manière suivante :

```
printf ("Nom %s date %d %d %d\n",
      anniversaire.nom,
      anniversaire.date->jour,
      anniversaire.date->mois,
      anniversaire.date->annee);
```

Tableaux de structure

On peut créer des tableaux de structures. Chaque élément du tableau sera une structure.
Exemple : On veut créer un répertoire téléphonique et on a besoin pour chaque personne de la structure suivante :

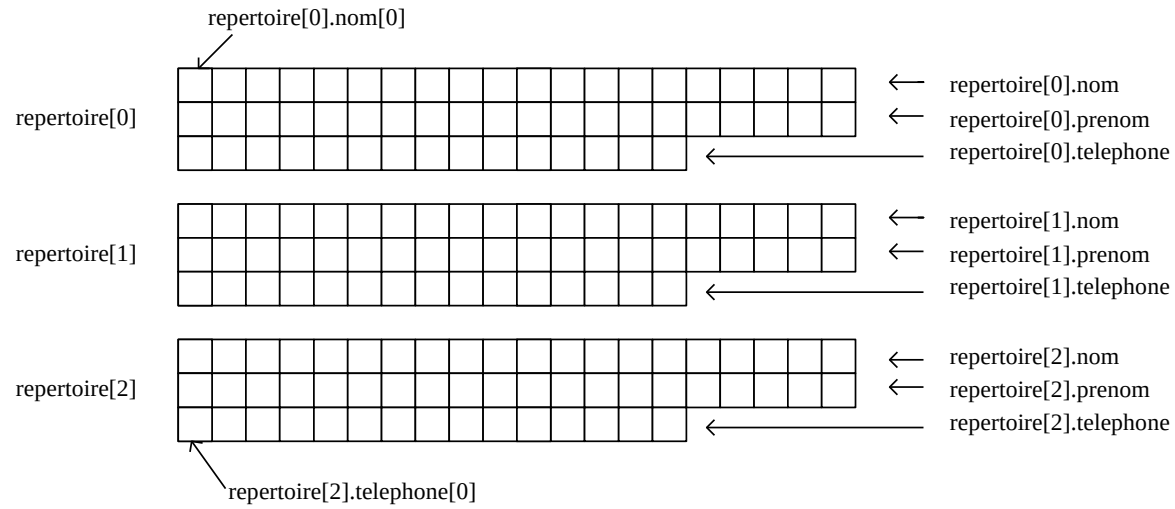
```
struct typePersonne {
      char nom[20];
      char prenom[20];
      char telephone[15];
};
```

Le répertoire téléphonique sera stocké dans le tableau suivant :

```
struct typePersonne repertoire[100];
```

Cette instruction déclare un tableau `repertoire` de 100 éléments. Chacun de ces éléments est une structure de type `typePersonne` qui sera accessible par un index ou par un pointeur. Chacune de ces structures regroupe 3 éléments qui sont des tableaux de type `char`.

Voici le diagramme de cette construction :



Les éléments d'un tableau de structures sont stockés en mémoire séquentiellement. La structure typePersonne étant composé de 55 octets

Pointeurs vers un tableau de structures

On peut utiliser des pointeurs sur les tableaux de structures. La déclaration d'un pointeur utilise l'opérateur devant le nom du pointeur :

```
struct typePersonne *pointeurPersonne ;
```

Ce pointeur pourra être utilisé pour pointer un élément du tableau répertoire défini ci-dessus.

```
pointeurPersonne = repertoire ;
```

Cette instruction initialise le pointeur pointeurPersonne à l'adresse du premier élément du tableau repertoire. On aurait aussi pu écrire :

```
pointeurPersonne = &repertoire[0] ;
```

pointeurPersonne->nom permet d'accéder au membre nom de l'élément pointé.

Incrémentation et décrémentation d'un pointeur sur un tableau de structures :

L'incrémentation d'un pointeur augmente le pointeur de la taille de la structure pointée.

```
pointeurPersonne = repertoire ;
```

Si après cette instruction, on exécute l'instruction suivante :

```
pointeurPersonne ++ ;
```

`pointeurPersonne` pointera sur le 2ème élément du tableau. Cette incrémentation augmente le pointeur de 55 octets, soit la taille de la structure.

De la même manière, la décrémentation d'un pointeur décrémente le pointeur de la taille de la structure pointée.

Exercice :

12.3 Créer un répertoire d'adresses à l'aide d'un tableau de structures comme ci-dessus.

Écrire une fonction saisiePersonne qui saisie un élément du tableau.

Écrire une fonction printPersonne qui imprime un élément du tableau.

Écrire le programme principal qui appelle les 2 fonctions précédentes pour initialiser le répertoire, et l'imprimer. Faire un système de menu : 1 pour ajouter une personne, 2 pour imprimer le répertoire et 0 pour sortir.

Les Unions

Les unions sont des nouvelles structures de données assez proches des structures. Elles permettent de stocker dans une variable des données de types différents. Par contre, elles ne peuvent contenir qu'une seule donnée à la fois.

En fait, las seule différence entre les unions et les structures est la manière d'organiser les champs en mémoire. Les structures stockent leurs champs séquentiellement et la taille d'une structure est donc la somme de la taille de chacun de ses champs, plus quelques octets éventuellement pour le cadrage sur une frontière de mot machine. Les unions stockent tous leurs champs à la même adresse, c'est à dire que les champs d'une union se recouvrent, ce qui explique qu'il ne puisse y avoir qu'une seule donnée à la fois dans une union. La taille d'une union est alors la taille du plus grand composant de l'union.

Les opérations possibles sur les unions sont les mêmes que les opérations sur les structures :

- affectation '='
- accès à un membre de l'union '.' ou '->'
- adresse d'une union '&'

Définition d'une union :

```
union nomUnion {
      membre1 ;
      ...
} variable ;
```

Comme pour les structures, on peut déclarer des variables à la définition de l'union ou seulement définir l'union et déclarer des variables plus loin dans le programme.

Exemple
Définition d'une union :

```
      union exemple {
            char car;
            int val;
      };
```

Déclaration d'une variable de type union :

```
      union exemple toto;

      toto.car = 'a';
      printf ("car : %c, val : %X\n", toto.car, toto.val);
      toto.val = 0;
      printf ("car : %c, val : %X\n", toto.car, toto.val);
```

Ces instructions donnent le résultat suivant :

```
car : a, val : CCCCCC61
car :  , val : 0
```

Dans la première ligne, un seul caractère à été initialisé dans l'entier et dans la deuxième ligne, le caractère n'est pas un caractère imprimable.

Structure et union
Une union peut être membre d'une structure tout comme une autre variable.
Exemple :

```
struct typeDonnee {
      int type ;
      union part {
            char car;
            int val;
      } donnee;
};
void printDonnee (struct typeDonnee d);
void main () {
      struct typeDonnee d1 = {0, 'a'};
      struct typeDonnee d2 = {1, 100};

      printDonnee (d1);
      printDonnee (d2);
}
void printDonnee (struct typeDonnee d) {
      if (d.type == 0)
            printf ("Caractere : %c\n", d.donnee.car);
      else
            printf ("Entier : %d\n", d.donnee.val);
}
```

Utilisation de typedef :
Le mot clé typedef permet de créer un synonyme pour un type de donnée existant. Le nouveau type défini peut être ainsi utilisé pour déclarer des variables.

Exemple :

typedef int entier ;

```
entier valeur ;
```

La variable déclarée de ce type a toutes les propriétés d'une variable de type int.

Utilisation de typedef pour les structures :
Le mot clé typedef permet de créer le synonyme d'une structure ou d'une union. Les instructions suivantes, par exemple, défininssent coord comme synonyme d'une structure ou d'une union.

```
typedef struct typeCoord {
      int x;
      int y;
} coord;

coord p1; /* On n'a plus besoin du mot clé struct */
```

est équivalent à :

```
struct typeCoord p2;
```

Dans la définition du nouveau type, on n'est pas obligé de donner un nom à la structure :

```
typedef struct {
      int x;
      int y;
} coord;
```

Exercice :
12.4 Soit la structure suivante :

```
struct typeDonnee {
      int type ;
      union part {
            char car;
            int val;
      } donnee;
};
```

Définir un tableau d'éléments de cette structure dont la taille maximum est fixée par une directive `define`.
Lire une expression arithmétique et entrer tous les éléments de l'expression dans le tableau.
Imprimer tous les éléments du tableau.
Les valeurs entières seront stockées dans le membre '`val`' de l'union et les opérateurs seront stockés dans le membre '`car`' de l'union.
Le membre type sera égal à 0 si la structure représente un opérateur et à1 si la structure représente un opérande.

13 Gestion de fichiers

Les fichiers sont utilisés pour stocker des données sur disques, disquettes, bandes magnétiques et aussi pour accéder à certains périphériques (clavier, écran, imprimante…). Les fichiers ne sont pas prévus en tant qu'objets spécifiques dans le langage C mais la librairie standard C définit des fonctions pour les manipuler.

Un fichier contient une suite d'octets qui doivent être interprétés par les applications qui les gèrent. Les fonctions de lecture et d'écriture de fichiers ainsi que le système d'exploitation, nous conduisent vers la distinction de 2 types de fichiers :

Fichier de type texte : les données sont une suite de caractères. Un fichier est composé de lignes qui se terminent par un ou plusieurs caractères signifiant *fin de ligne*. Une ligne n'est pas une chaîne de caractères ; il n'y a pas de \0 terminal. Lorsqu'un fichier est utilisé en mode texte, une traduction s'effectue entre le caractère de fin de ligne du langage C ,'\n', et les caractères utilisés par le système d'exploitation comme terminateur de ligne. Sur les systèmes DOS et Windows, le retour à la ligne est représenté par 2 caractères (LF et CR). Lorsqu'on écrit '\n' sur un fichier en mode texte, le système écrit ces 2 caractères. Quand on lit un fichier en mode texte, ces 2 caractères lus sont traduits par le caractère '\n'. Sur les systèmes UNIX, le caractère de fin de ligne est '\n', il n'y a pas de traduction ; il n'y a pas de différentiation entre fichiers binaires et fichiers texte.

Fichier de type binaire : Toutes les informations sont écrites et lues telles quelles, sans aucune séparation entre les lignes et sans caractère de fin de ligne. Les caractères '\0' et '\n' n'ont pas de signification particulière. En C, on crée des fichiers binaires quand on écrit des valeurs sous la forme de leur représentation en mémoire (ex 1 entier = 32 bits). Ces données doivent être relues suivant le même format si on veut retrouver les mêmes valeurs. Le format binaire permet de gagner de la place par rapport à un format ASCII (exemple : un entier est toujours sur 32 bits (soit 4 caractères) quelque soit sa valeur en format binaire mais converti en ASCII, il occupe une place variable de 1 à 10 caractères « 4294967296 »).

La principale différence entre un fichier binaire et un fichier texte est l'interprétation des caractères de fin de ligne sous les systèmes Windows. Sur les systèmes Unix, il n'y a pas d'option qui différencie les 2 modes.

Le fichier ***stdio.h*** définit les prototypes des fonctions d'entrée/sortie pour la création et la manipulation des fichiers de données.

Fichiers prédéfinis
Il existe des fichiers prédéfinis et utilisés de façon implicite dans des fonctions comme *scanf*, *printf* ou *gets*.

Fichier	Description	Périphérique	Fonctions
stdin	entrée standard	clavier	scanf, gets
stdout	sortie standard	écran	printf, puts
stderr	fichiers d'erreurs	écran	printf, puts

Type d'accès
Les fonctions de lecture/écriture permettent 2 techniques d'accès aux fichiers :

. accès séquentiel : lecture de tous les caractères depuis le premier

. accès direct : on se place directement sur l'information souhaitée.

On peut utiliser ces 2 techniques sur tous les fichiers et même mélanger les 2 techniques d'accès sur un même fichier. Les fonctions de lecture et d'écriture dans un fichier gère un pointeur de données dans le fichier qui permet de déterminer où sera écrite ou lue la prochaine donnée.

Accès au fichier
Les fonctions suivantes permettent les opérations de lecture et d'écriture dans les fichiers.

Pointeur de fichier

```
FILE *descr ; /* Pointeur sur un descripteur de fichier */
```

Signifie que fichier est un pointeur sur un objet de type FILE. Le type FILE définit un descripteur de fichier. Un descripteur de fichier contient toutes les informations utiles concernant le fichier : son nom, son emplacement, son mode d'ouverture, positionnement courant dans le fichier.
L'identificateur FILE désigne un modèle de structure défini dans le fichier stdio.h (par une instruction typedef, d'où l'absence du mot struct).
Il faut déclarer une variable de ce type pour chaque fichier à accéder dans le programme.
`stdin, stdout et stderr` sont des descripteurs de fichiers prédéfinis.

Ouverture de fichier

Un fichier peut être accédé après avoir été ouvert par la fonction fopen :

```
FILE *fopen(char *nomfichier, char *mode) ;
```

nom_fichier : chaîne de caractères (ex "essai.dat"). Le fichier est cherché par rapport au répertoire courant d'exécution. On peut donner le nom complet du fichier.
(ex "C:\monRépertoire\essai.dat").

mode : chaîne de caractères qui est composé des caractères suivants :

mode	Signification
" r "	Ouverture d'un fichier existant en lecture seulement.
" w "	Création et ouverture d'un fichier en écriture seulement. Si le fichier n'existe pas, il est créé et s'il existe, il est détruit et remplacé par le fichier créé.
" a "	Ouverture d'un fichier existant pour lui ajouter des enregistrements en fin de fichier. Le fichier est créé s'il n'existe pas au préalable.
" r+ "	Ouverture en lecture et écriture d'un fichier déjà existant.
" w+ "	Création d'un fichier et ouverture en lecture-écriture. Si le fichier nom_fichier existe déjà, il est détruit et remplacé par le fichier créé.
" a+ "	Ouverture en lecture et écriture d'un fichier auquel on peut ajouter des enregistrements. Si le fichier nom_fichier n'existe pas au préalable, il est alors créé.

Rajouter **b** dans la chaîne de caractères mode **si le mode de lecture est binaire.**
Par défaut, un fichier est considéré en mode texte.

Les opérations de lecture dans un fichier sont autorisées pour les modes : r, r+, w+ et a+.
Les opérations d'écriture dans un fichier sont autorisées pour les modes : w, a, w+ et a+.

La fonction `fopen` initialise un descripteur de fichier avec le nom du fichier et le mode d'ouverture et renvoie un pointeur sur ce descripteur de fichier ou NULL si le fichier n'a pas pu être ouvert. Un fichier ne peut pas être ouvert dans le cas où on n'a pas les droits d'accès, si le fichier n'existe pas en ouverture en lecture, si on n'a pas le droit de créer un fichier.
La fonction initialise aussi le positionnement courant en début de fichier dans le descripteur de fichier. Ce qui permettra de lire le premier octet du fichier lorsqu'on fera la première lecture dans le fichier.
Après l'appel de la fonction fopen, on doit toujours tester si le résultat est NULL avant de lire ou d'écrire dan le fichier.

Exemple :

```
FILE *fichierEleve ;
fichierEleve = fopen ("listeEleve.txt", "r");
```

Fermeture de fichier

```
fclose(FILE *descr) ;
```

Le fichier doit être fermé lorsqu'on a fini de l'utiliser dans le programme. La fonction fclose permet de vider la mémoire tampon et de libérer l'espace mémoire utilisé pour le descripteur de fichier. En mode écriture, si le fichier n'est pas fermé avant de sortir du programme, le fichier ne sera pas complet.
Exemple :

```
FILE *fichierEleve ;
fichierEleve = fopen ("listeEleve.txt", "r");
if (fichierEleve != NULL) {
      fclose (fichierEleve); //ATTENTION : fermer si fopen != 0
}
```

Fin de fichier

```
int feof(FILE *descr) ;
```

Cette fonction teste si la position courante dans le fichier est en fin de fichier. Il rend la ***valeur*** *1 (VRAI) si c'est la fin de fichier et 0 (FAUX) dans le cas contraire.*

Position courante dans le fichier :

Toutes les fonctions de lecture et d'écriture dans un fichier modifient la position courante dans le fichier. La position courante dans le fichier est stockée dans le descripteur de fichier. A l'ouverture du fichier, la position courante est initialisé sur le premier caractère (ou octet) du fichier. La position courante dans le fichier permet de savoir à partir de quel caractère (ou octet) commencera la prochaine lecture ou après quel caractère sera écrit le prochain caractère (ou octet) dans le fichier.

Une fonction de lecture dans un fichier provoque le déplacement de la position courante sur le caractère (ou octet) juste après le dernier caractère lu. Une fonction d'écriture dans le fichier déplacera la position courante juste après le dernier caractère écrit.

Lecture de chaîne de caractères :

`char *fgets (char *s, int max, FILE *descr)` ;

Cette fonction lit la ligne courante dans un fichier (y compris le caractère de fin de ligne) et la stocke dans le tableau de caractère s ; cette fonction lira au maximum max-1 caractères). La chaîne de caractères s qui en résulte est terminée par '\0' (c'est pour garder un octet pour ce caractère que la fonction lit seulement max-1 caractères au maximum). Cette fonction permet de lire une chaîne de caractères en s'assurant de la taille maximum de la chaîne lue. Le résultat est NULL si la fonction n'a pas réussi à lire une chaîne.

Exemple :

Lecture de toutes les lignes d'un fichier et impression à l'écran :

```
FILE *fichNom;
char ligne [LG] ;
fichNom = fopen ("listeNom.txt", "r");
if (fichNom == NULL) {
   printf ("Le fichier listeNom.txt n'est pas trouve\n") ;
} else {
   fgets(ligne, LG, fichNom) ;/*Lecture 1ere ligne du fichier */
   while (!feof (fichNom)) {
      printf ("%s", ligne);   /*Affiche la ligne lue dans fichier */
      fgets(ligne, LG, fichNom) ;  /*Lecture de la ligne suivante */
   }
   fclose (fichNom);
}
/* Ce programme est la commande type de DOS */
```

`int fscanf (FILE *descr, char *format, listeadresses) ;`

Cette fonction a le même comportement que la fonction scanf mais les données sont lues dans le fichier au lieu d'être lues au clavier. Cette fonction rend le nombre de variables lues et stockées. Le résultat sera inférieur aux nombres de variables à lire s'il y a eu une erreur.

```
fscanf (fich, "%d ", &valeur);
```

Ecriture de chaîne de caractères :

`int fputs (char *s, FILE *descr);`

Cette fonction écrit la chaîne s dans le fichier sans ajouter de \n (contrairement à puts) et a pour résultat le dernier caractère écrit ou EOF (constante définie dans stdio.h qui vaut –1) si erreur.

Exemples :

```
char message[] = "une ligne de texte ";
fputs (message, fich);
fputs ("fin ", fich);
```

`int fprintf (FILE *descr, char *format, listearguments) ;`

Cette fonction écrit la chaîne de caractères définie par le format et la liste d'arguments exactement comme la fonction printf mais la chaîne résultat est écrite dans le fichier au lieu d'être affiché à l'écran. La fonction rend le nombre d'octets écrits, ou EOF si erreur. Les \n sont transformés en CR/LF si le fichier a été ouvert en mode texte (spécifique DOS).

Exemple : `fprintf (fich, " total %8.2f ", total);`

Exemple :

```
#define LGL 100

char ligne[LGL] ;
FILE *entree, *sortie ;
entree = fopen ("fichier1.txt", "r"); // Ouverture en mode lecture
sortie = fopen ("fichier2.txt", "w"); // Ouverture en mode créationa
if (entree != NULL && sortie != NULL) {
   fgets (ligne, LGL, entree) ; /*Lecture de la 1ère ligne*/
   while ( !feof (entree)) {
      fputs (ligne, sortie) ; /* Ecriture de la ligne lue */
      fgets (ligne, LGL, entree)); /*Lecture de la ligne suivante */
   }
   fclose (entree) ;
   fclose (sortie) ;
}
```

Lecture du caractère courant dans le fichier.

```
char fgetc(FILE *descr);
```

Ecriture d'un caractère dans le fichier.

```
char fputc(char c, FILE *descr) ;
```

Les fonctions de lecture et d'écriture binaire dans un fichier permettent de traiter un certain nombre d'octets sans se soucier du contenu de ces octets. Ces fonctions ne sont pas conçues pour être utilisées avec des fichiers composés uniquement de texte.

Ecriture binaire dans un fichier

```
int fwrite (void *bloc, int taille, int nb, FILE *descr) ;
```

Cette fonction écrit `nb*taille` octets, le premier octet à l'adresse '`bloc`'. Si le nombre rendu est différent de nb, il y a eu erreur.

Exemple :

```
FILE *fich ;
int valeur ;
fich = fopen ("listeValeur.dat", "w");
valeur = 10;
fwrite (&valeur, sizeof(int), 1, fich) ; /*Ecriture de 1 entier */
```

Lecture binaire dans un fichier

```
int fread(void *bloc, int taille, int nb, FILE *id) ;
```

Cette fonction lit nb*taille octets qui seront rangés en mémoire à l'adresse bloc. La fonction rend le nombre d'éléments effectivement lus (<nb si fin de fichier), 0 si erreur.

Exemple :

```
FILE *fich ;
int valeur ;
fich = fopen ("listeValeur.dat", "r");
fread (&valeur, sizeof(int), 1, fich) ; /*Lecture de 1 entier */
```

Exemples :

1. Lecture d'un fichier texte :

```
   /*
 * Afficher à l'écran le contenu d'un fichier texte
 */
#include <stdio.h>
#include <stdlib.h>
#define LGL 81       /* Longueur max d'une ligne de texte    */
void main () {
   char ligne [LGL];
   char nomFichier [LGL];
   FILE *fich;
   /* Ouverture du fichier a lire*/
   printf("Nom du fichier a lire: ");
   fgets (ligne, LGL, stdin);
   sscanf (ligne, "%s", nomFichier);
   fich = fopen (nomFichier, "r");
   if ( fich == NULL ) {
     printf (Ouverture impossible fichier %s\n", nomFichier);
   } else {
     printf ("Fichier %s \n", nomFichier);
     fgets (ligne, LGL, fich); // Lecture 1ère ligne
     while (!feof(fich)) {     // Tant que non fin de fichier
       printf ("%s", ligne);   // Imprimer ligne à l'écran
       fgets (ligne, LGL, fich); // Lecture ligne suivante
     }
     fclose (fich);
   }
}
```

2. Création séquentielle d'un fichier binaire :

```
/* Lecture d'entiers au clavier et écriture de ces entiers dans
   un fichier. Le programme s'arrête lorsque la valeur '0' est lue */
#include <stdio.h>
#define MAXLG 80
#define MAXNF 80
int saisir_entier (void);
void main()
{
   char nom_fichier[MAXNF];
   char ligne[MAXLG];
   int n;
   FILE *fich;

   printf("Nom du fichier a creer: ");
   fgets (ligne, MAXLG, stdin);
   sscanf(ligne, "%s", nom_fichier);
   fich = fopen(nom_fichier, "wb");
   n = saisir_entier ();
   while (n != 0) {
      fwrite(&n, sizeof(int), 1, fich); /*Ecriture*/
      n = saisir_entier ();
   }
   fclose(fich);
}
int saisir_entier (void){
   char ligne[MAXLG];
   int val, n;
   do {
      printf("Introduire un nombre entier: ");
      fgets (ligne, MAXLG, stdin);  // Saisir une ligne au clavier
      n = sscanf(ligne, "%d", &val);  // lire un entier dans la ligne
      if (n != 1) {
         puts ("Erreur");
      }
   } while (n!=1);
   return (val);
}
```

3. Lister un fichier binaire

```
/* Lit une suite d'entiers dans un fichier et l'affiche à l'écran */
#include <stdio.h>
void main()
{
   char nom_fichier[21];
   int n;
   FILE *fich;
   printf("Nom du fichier a lire: ");
   scanf("%20s", nom_fichier);
   fich = fopen(nom_fichier, "rb");
   if (fich != NULL) {
      fread(&n, sizeof(int), 1, fich);
      while (!feof (fich)) {
         printf("%d\n",n);
         fread(&n, sizeof(int), 1, fich);
      }
      fclose(fich);
   } else {
      printf ("Impossible d'ouvrir le fichier %s\n", nom_fichier);
   }
```

Accès direct

La fonction fseek permet d'agir sur la position courante dans le fichier, sans devoir lire ou écrire les données qui précèdent. On peut ainsi accéder directement une donnée qui n'est pas en début de fichier.

```
int fseek(FILE *descr, long nb, int mode) ;
```

Cette fonction déplace le pointeur de données du fichier dont le descripteur est **desc**, de **nb** octets à partir :

Mode	**Positionnement**
SEEK_SET	Nb octets à partir du début de fichier
SEEK_CUR	Avance de nb octets à partir de la position courante
SEEK_END	Recule de nb octets à partir de la fin de fichier.

La fonction retourne 0 si tout s'est bien passé.

Cette fonction seulement si on connaît la taille des données dans le fichier (Il est impossible de se positionner directement sur une ligne donnée d'un texte si on ne connaît pas la longueur de chaque ligne).

Exemple : Dans un fichier contenant des entiers, le programme se positionne sur le n'ième entier dans le fichier (n est saisi au clavier ainsi que le nom de fichier) .

```
/* Accès direct en lecture sur un fichier existant */
/* Lecture de n : numéro de l'entier à lire dans le fichier */
/* Lecture de l'entier numéro n dans le fichier et affichage */
#include <stdio.h>
#define MAXLG 80
#define MAXNF 80
int saisir_entier (void);
void main()
{
   char nom_fichier[MAXNF];
   char ligne[MAXLG];
   int n, num;
   FILE *fich;

   printf("Nom du fichier a consulter: ");
   fgets (ligne, MAXLG, stdin);
   sscanf(ligne, "%s", nom_fichier);
   fich = fopen(nom_fichier, "rb");
   if (fich != NULL) {
      num = saisir_entier ();
      while (num != 0) {
         /* On se positionne sur le num ième entier */
         fseek (fich, sizeof(int)*(num-1), SEEK_SET);
         fread (&n, sizeof(int), 1, fich);
         if (!feof (fich)) {
              printf("valeur = %d\n", n);
         } else {
              puts ("Fin de fichier");
         }
         num = saisir_entier ();
      }
      fclose(fich);
   } else {
      printf ("Impossible d'ouvrir le fichier %s\n", nom_fichier);
    }
}
```

```
int saisir_entier (void){
      char ligne[MAXLG];
      int val, n;
      do {
          printf("numero de l'entier recherche: ");
            fgets (ligne, MAXLG, stdin);    // Saisir une ligne au clavier
            n = sscanf(ligne, "%d", &val);  // lire un entier dans la ligne
            if (n != 1) {
                  puts ("Erreur");
            }
      } while (n!=1);
      return (val);
}
```

Exercices :

13.1 Écrire un programme qui crée un fichier texte et qui le remplit à partir de lignes de texte saisies au clavier (La saisie s'arrête sur la ligne vide)

13.2 Lire un fichier texte (par exemple le fichier créé dans l'exercice précédent) et créer un nouveau fichier en remplaçant tous les espaces par des '-'.

13.3 Lire un fichier composé de nombres à virgules flottantes, afficher la somme et la moyenne de ces valeurs.

13.4 Soit un fichier texte client.txt qui contient la liste des articles commandés et le prix unitaire pour un client. Ecrire la facture dans un fichier nommé facture.txt. La facture contiendra une ligne par article avec le nom du produit, la quantité commandée, le prix unitaire et le prix total (quantité*nombre). La dernière ligne contiendra le prix total de la facture.

Exemple de fichier client.txt :

```
Imprimante      2     321.50
Clavier         5     56.75
Ecran           1     451,00
```

13.5 Création d'un fichier des étudiants.

Un menu permettra de choisir une des commandes suivantes :

1 => Saisir les informations sur un étudiant et les écrire dans le fichier.

2 =>Saisir un numéro d'étudiant et l'afficher (ex afficher le 3ème étudiant)

0 => Sortie du programme

La structure suivante sera utilisée pour stocker les informations sur un étudiant :

```
struct {
  char nom [LGNOM];
  char prenom [LGNOM];
  int age;
} etudiant ;
```

Annexe 1 : Fonction scanf

```
int scanf ("format", adresseVariable1, adresseVariable2, …);
```

Description :

La fonction *scanf* permet de faire une lecture formatée du flux standard d'entrée (le clavier par défaut).
Elle lit les caractères en entrée et les interprète en concordance avec les spécifications de format décrites dans la chaîne *format*, et place les résultats dans les arguments.
Pour pouvoir retourner les valeurs ainsi saisies, **les arguments doivent être obligatoirement des pointeurs *(adresses des variables)***.
L'opérateur & permet d'obtenir l'adresse d'une variable : &toto.

Valeur retournée :

le nombre de valeurs convenablement introduites ou *EOF* (-1)en cas d'erreur.

Remarques :

Les informations tapées au clavier sont d'abord mémorisées dans un tampon avant d'être traitées par *scanf*. La zone tampon est passée à la fonction après la frappe de la touche *return* au clavier.
La chaîne de format ne doit comporter que des spécifications de format, tout autre caractère peut amener le programme à se comporter curieusement.

Spécificateurs de format :

Ils sont introduits par le caractère % et se terminent par le caractère de type de conversion suivant le format suivant :

% [largeur] [modificateur] type

largeur : elle précise la nombre de caractères n qui seront lus. On peut en lire moins si l'on rencontre un séparateur (espace, tabulation, retour chariot ...) ou un caractère invalide.

modificateur : Il précise la taille de l'objet recevant la valeur.

Modificateur	l'objet recevant est
h	un entier de type short int (d,i,o,u,x)
l	un entier de type long int (d,i,o,u,x) un réel de type double (e,f)
L	un réel de type long :double (e,f,g)

type : type de l'objet pointé par arg.

type	Type de l'objet pointé
d	signed int exprimé en décimal
o	signed int exprimé en octal
u	unsigned int exprimé en décimal
x	int (signed ou unsigned) exprimé en hexadécimal
f,e,g	réel
c	largeur non spécifiée ou égale à 1 : caractère

Exemple :

```
#include <stdio.h>

void main() {
  int i,j;
  double d;

  printf("entier: ");
  scanf("%d", &i);
  printf("2 entiers et 1 double: ");
  scanf("%d%d%lf", &i, &j, &d);
}
```

Problèmes rencontrés avec la fonction scanf :

***Cas 1* :** la fonction scanf trouve les valeurs avec les bons types tels qu'ils étaient décrits et il n'y a rien d'autre dans la ligne. C'est un cas de bon fonctionnement et ce cas ne pose pas de problème.

***Cas 2* :** les valeurs trouvées sont du bon type mais il n'y en a pas assez : `scanf` reste en attente des valeurs. (exemple : si le format est `"%d%d" et si` l'utilisateur a entré un seul entier et pas d'autre valeur avant de frapper la touche 'entrée'. Il faut entrer un autre entier au clavier pour que le programme continue).

***Cas 3* :** les valeurs trouvées sont du bon type mais il y a des valeurs en plus dans la ligne. (exemple : l'utilisateur entre 3 entiers dans la ligne au lieu de 2). Dans ce cas, les valeurs en trop restent dans la mémoire tampon et c'est le prochain `scanf` qui les lira. Ce qui pose un problème car le prochain `scanf` n'attendra pas forcément un entier.

***Cas 4* :** les valeurs entrées ne sont pas toutes du bon type. `scanf` arrête sa recherche dans le buffer d'entrée dès qu'il trouve une valeur qui ne correspond pas au type attendu. (exemple : on entre un entier et un caractère au lieu de 2 entiers, le prochain `scanf` lira le caractère resté dans la mémoire). Les caractères qui ne correspondent pas au format décrit dans l'instruction `scanf` resteront dans la mémoire tampon et le programme aura un comportement inattendu. Il faut lire le résultat rendu par la fonction `scanf` pour savoir si `scanf` a trouvé les bonnes valeurs.

Solution :

L'instruction **`fflush(stdin)`** permet de vider la mémoire tampon.

Exemple :

```
#include <stdio.h>
void main ()
{
     int n, i, j;
     i = j = 0;
     printf ("Entrer 2 entiers : ");
     n = scanf ("%d %d", &i, &j); /* n est égal à 0, 1 ou 2*/
     printf("Nombre de valeurs correctes : %d, i : %d, j :%d\n",
             n,i,j);
     printf ("Entrer 2 entiers : ");
     n = scanf ("%d %d", &i, &j); /* n est égal à 0, 1 ou 2*/
     printf("Nombre de valeurs correctes : %d, i : %d, j :%d\n",
             n,i,j);
     }
}
Entrer 2 entiers : a
Nombre de valeurs correctes : 0, i : 0, j : 0
Entrer 2 entiers : Nombre de valeurs correctes : 0, i : 0, j : 0
```

Si on rajoute `fflush (stdin)` après le premier appel à `scanf` :

```
#include <stdio.h>
void main ()
{
     int n, i, j;
     i = j = 0;
       printf ("Entrer 2 entiers : ");
       n = scanf ("%d %d", &i, &j); /* n est égal à 0, 1 ou 2*/
       fflush(stdin);
       printf ("Nombre de valeurs correctes : %d, i : %d, j : %d\n",
                n,i,j);
       printf ("Entrer 2 entiers : ");
       n = scanf ("%d %d", &i, &j); /* n est égal à 0, 1 ou 2*/
       printf("Nombre de valeurs correctes : %d, i : %d, j :%d\n",
                   n,i,j);

     }
}
Entrer 2 entiers : a
Nombre de valeurs correctes : 0, i : 0, j : 0
Entrer 2 entiers : 12 34
Nombre de valeurs correctes : 2, i : 12, j : 34
```

Annexe 2 : Fonction printf

```
printf  ("format", expr1, expr2, expr3) ;
```

Description :

Elle permet l'écriture formatée sur le flux standard de sortie *stdout* (l'écran par défaut).
La chaîne de caractères format peut contenir à la fois :

. Des caractères à afficher,

. Des spécifications de format.

Il devra y avoir autant d'arguments à la fonction *printf* qu'il y a de spécifications de format.

Valeur retournée :

Le nombre d'octets effectivement écrits ou la constante *EOF* (-1) en cas d'erreur.

Spécificateurs de format :

Ils sont introduits par le caractère **%** et se terminent par le caractère de type de conversion suivant la syntaxe suivante :

```
% [drapeaux] [largeur] [.precision] [modificateur] type
```

drapeaux : il précise la justification et le préfixe.

Drapeaux	Signification
rien	Justifié à droite et complété à gauche par des espaces
-	Justifié à gauche et complété à droite par des espaces
+	Les valeurs numériques sont précédées du signe + ou -
#	Sans effet sur les types c, s, d, i, u Type o : '0' précédera la valeur Type x, X : : 0x ou 0X sera placé devant la valeur Type e, E, f : le point décimal sera toujours affiché Type g, G : affichage du point décimal et un zéro à droite (si nécessaire)

largeur : elle précise le nombre de caractères *n* qui seront affichés.

Si la valeur à afficher dépasse la taille de la fenêtre ainsi définie, C utilise quand même la place nécessaire (une valeur ne peut pas être tronquée). Le point est compris dans la largeur pour les variables de type float.

Largeur	Effet
n	Affiche n caractères complétés éventuellement par des espaces
0n	Affiche n caractères complétés éventuellement à gauche par des 0
*	L'argument précédent la valeur à afficher fournit la largeur

précision : elle précise pour :

un entier : le nombre de chiffres à afficher

un réel : avec %f : le nombre de chiffres de la partie décimale à afficher (avec arrondi) et avec %g : le nombre de chiffres à afficher.

les chaînes de caractères : le nombre maximum de caractères à afficher.

.precision	Effet
rien	Précision par défaut d,i,o,u,x : pas de chiffre après la virgule e, E, f : 6 chiffres pour la partie décimale
.n	n caractères au plus
*	Précision dynamique, argument qui précède la valeur à afficher

modificateur : Il précise le type (obligatoire pour les types short, long, double)

Modificateur	Nécessaire pour	Utiliser avec
h	un entier de type short	d, i, o ,u, x, X
l	un entier de type long	d, i, o, u, x, X
L	un réel de type long double	e,E,f,g,G

type : type de conversion de l'argument.

Type	Format de la sortie
d *ou* i	entier décimal signé
o	entier octal non signé
u	entier décimal non signé
x	entier hexadécimal non signé
X	entier hexadécimal non signé en majuscules
f	réel de la forme [-]dddd.ddd
e	réel de la forme [-]d.ddd e [+/-]ddd
E	comme e mais l'exposant est la lettre E
g	format intelligent : e ou f suivant la précision et la valeur
G	comme g mais l'exposant est la lettre E
c	caractère
s	affiche les caractères jusqu'au caractère nul '\0' ou jusqu'à ce que la précision soit atteinte
p	pointeur

Exemple :

```
#include <stdio.h>

void main() {
  int nbre = 5;
  float prix = 12;
  float result = prix * nbre;

  printf("Bonjour\n");
  printf("Non formate : Nombre %d prix %f Total %f\n",
                          nbre, prix, result);
  printf("Format       : Nombre %2d prix %.2f Total %.2f\n",
                          nbre, prix, result);
}
/*-- résultat de l'exécution ---------------------------------
Bonjour
Non formate : Nombre 5 prix 12.000000 Total 60.000000
Formate     : Nombre  5 prix 12.00 Total 60.00
------------------------------------------------------------------*/
```

Exemple d'utilisation des formats numériques :

```
#include <stdio.h>

void main () {
     printf("1|%8d|%8d|%+8d|\n",12345,-12345,12345);
     printf("2|%08d|%-8d|%-08d|\n",12345,12345,12345);
     printf("3|%6x|%06X|%#06X|%-#6X|\n",255,254,253,255);
     printf("4|%10f|%10.2f|%010.2f|\n",1.234,1.234,1.456);
     printf("5|%g|%f|%8.3g|%10g|%10.2g|\n",1.234,1.234,1.234,
            12222222222222.234,12222222222222.234);

}
Résultat :
1|   12345|  -12345|  +12345|
2|00012345|12345   |12345   |
3|    ff|0000FE|0X00FD|0XFF  |
4|  1.234000|      1.23|0000001.46|
5|1.234|1.234000|    1.23|1.22222e+013|  1.2e+013|
```

Caractères spéciaux le plus souvent utilisés :

Ce sont des caractères qu'on peut introduire dans le texte.

Ordre	Signification
\a	Sonnerie
\b	Retour arrière
\n	Retour à la ligne
\t	Tabulation
\\	\ (on veut imprimer ce caractère)
\ "	" (on veut imprimer ce caractère)
%%	%

Fonction puts
La fonction puts permet aussi d'afficher du texte à l'écran, mais pas de variable. Elle reçoit comme argument une chaîne de caractères et l'envoie sur l'écran en ajoutant automatiquement un retour à la ligne.

Exemple :

```
puts ("Bonjour") ;
```

aura le même résultat que l'instruction `printf ("Bonjour\n") ;`
Attention : puts rajoute '\n' à la fin de la chaîne de caractères

Exercices :

1. Saisir une date est une heure sous forme de 5 valeurs entières :
   - jour, mois, année, heure, minutes

   et imprimer le résultat formaté :

```
Entrer jour mois annee heure minutes : 18 1 1 12 9
Date : 18/01/01, heure : 12:09
```

2. Saisir un caractère et imprimer sa valeur en ASCII, décimal, octal et hexadécimal. Saisir une valeur entière et donner sa valeur en octal et en hexadécimal

```
Entrer un caractere : H
Caractere H, decimal 72, octal 0110, hexadecimal 0X48

Entrer un nombre entier : 120
octal 0170, hexadecimal 0X78
```

3. Saisir 3 valeurs à virgule flottante et imprimer les 3 valeurs et leur somme sous la forme d'une addition.

```
Entrer 3 valeurs : 35.4 22 1074.12

     35.40
 +   22.00
 + 1074.12
----------
   1131.52
```