Cours Langage C

Année 2015

Chapitre 1 : Généralité sur le langage C

Historique

Langage de programmation développé en 1970 par Dennie Ritchie aux Laboratoires Bell d'AT&T.

Il provient de deux langages :

- BPCL développé en 1967 par Martin Richards.
- B développé en 1970 chez AT&T par Ken Thompson.

Devant la popularité du C, l'American National Standard Institut charge en 1983 le comité X3J11 de standardiser le langage C.

Structure générale d'un programme en C

Exemples

Exemple1 : programme qui affiche Bonjour tout le monde

```
#include <stdio.h>
/* ce programme affiche à l'écran "Bonjour tout le monde« */
main() {
        printf("Bonjour tout le monde\n");
}
```

Exemple2 : programme qui affiche la somme de deux nombres entiers

```
#include <stdio.h>
/* ce programme nous permet de calculer la somme de deux
nombres entiers; a et b sont saisis au clavier*/
main() {
        /*déclaration des variables*/
        int a;
        int b;
        int somme;

        /*saisie des données*/
        printf("Entrer la valeur de a: \n");
        scanf("%d", &a );
        printf("Entrer la valeur de b: \n");
        scanf("%d", &b );

        somme=a+b;
        printf("La somme de %d et %d est: %d\n " , a, b, somme);
return0
}
```

Chapitre 2 : Les types de bases des variables

Les déclarations des variables sont de la forme:

type nom_variable [= <valeur>] ;

Elles donnent la liste des variables à utiliser, et indiquent leur type ainsi que leur valeur initiale éventuelle.

Les types de variables peuvent être :

- Des caractères uniques
- Des nombres entiers
- Des nombres réels (flottants)

Le type caractère unique

Une variable de type caractère est déclaré par le mot- clé **char.** Elle est codée sur **8bits** comme un caractère utilisé sur une machine (code ASCII) et ne contient qu'un seul caractère.

Exemple :

```
char  c1='e'; // déclaration d'une variable c1 de type char
              //à laquelle on affecte la valeur 'e'

char  c2=101; //c2 correspond également au caractère  'e'

char  c3='P'; ou char c3; // déclaration d'une variable c3 de type caractère
                  c3='P'; // on initialise  la valeur 'P' dans c3
```

L’affichage avec printf utilise après le symbole %, le suffixe **c.**

Exemple :

```
# include <stdio.h>
main (void)
      {
            char car1=’E’ ;
            char car2= ‘e’ ;
            char car3=69;
            char car4=101;
            printf (”les caractères  sont : car1 : %c\t car2 :” ”%c\ n”, car1, car2) ;

             /* modification de variable car avec la code ASCII de E et e */
            printf(”la modification des  : car3: %c\t  car4  %c\n”, car3, car4) ;
      }
```

code	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	NP	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	SP	!	”	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	-	DEL		

TAB. 1.1 – Codes ASCII en décimal

Il existe un certain nombre de caractères particuliers dont les principaux sont représentés dans le tableau suivant:
Tableau 2 : Quelques caractères spéciaux

CARACTERE	SEMANTIQUE	CODE ASCII
‘\n’	Retour à la ligne	10
‘\t’	Tabulation horizontale	9
‘\v’	Tabulation verticale	11
‘\b’	Retour arrière (Backspace)	8
‘\’ ’	Apostrophe	39
‘\ ’	Guillemet	34
\?	Point d’interrogation	63
\\	Backslash	92
\f	Saut de page	12
\0	Caractère nul	0

Ces caractères ont une signification particulier pour le langage C. Par exemple le caractère « guillemet » doit être utilisé dans les chaînes de caractères (la séquence ‘\guillemet ‘ permet de représenter ce caractère dans une chaîne).

De même, le caractère « ? » est utilisé dans les trigraphes qui sont des séquences de trois caractères permettant de désigner les caractères # [ ] \ ^ { } | ~. Le tableau 3 détaille les trigraphes disponibles.

Tableau 3: les 9 trigraphes disponibles

TRIGRAPHE	SIGNIFICATION	CODE ASCII
??=	# (dièse)	35
??(	[(crochet ouvrant)	91
??)	] (crochet fermant)	93
??/	Backslash	92
??’	^ (circonflexe ou chapeau)	94
??<	{ (accolade ouvrante)	123
??>	} (accolade fermante)	125
??!	|	124
??-	~ (tilda)	

Les chaînes de caractères sont vues comme un pointeur sur des caractères et de type char*

Exemple :

```
char*chaine=  "Bonjour tout le monde !"  ;// une chaine de caractère
                                         // noter l'utilisation du double
```

Le type « nombre entier »

On utilise le mot clé **int**. Dans le cas le plus général, on rencontre souvent les types de variables : short int, int, long int qui sont respectivement codés sur 16 bits, 16bits ou 32 bits et 32 bits.

Exemples::

```
int Nmax=15; // la variable Nmax est initialisée par la valeur 15
```

```
short int a; // la variable a est codée sur 16 bits
int b; // la variable b est codée sur 16bits ou 32 bits
long int c; //la variable c est codée sur 32 bits
long long int d; // la variable d est codée sur 64 bits
```

Le tableau suivant regroupe les types des entiers standards,
la taille mémoire utilisée et l'intervalle des valeurs possibles:

Type	Taille mémoire	Intervalle de valeurs
short short int signed short int	2 octets	$[-2^{15} ; 2^{15}-1]$
unsigned short int	2 octets	$[0; 2^{16}-1]$
int signed int	2 ou 4 octets	$[-2^{15} ; 2^{15}-1]$ ou $[-2^{31}; 2^{31}-1]$
unsigned int	2 ou 4 octets	$[0; 2^{16}-1]$ ou $[0; 2^{32}-1]$
long long int signed long	4 octets	$[-2^{31}; 2^{31}-1]$
unsigned long	4 octets	$[0; 2^{32}-1]$
long long (*)	8 octets	$[-2^{63} ; 2^{63}-1]$
Unsigned long long (*)	8 octets	$[0; 2^{64}-1]$

cas des constantes entières

En système de numération, on peut représenter une constante entière :

- en décimale (base 10),
- en octale (base 8),
- en hexadécimale (base 16).

Trois notations peuvent être distinguées ces représentations:

- Base 10: écriture usuelle; exemple :445
- Base 8: on commence par 0 (zéro) suivi de chiffres octaux;

exemple : 04701 qui vaut 2497 en décimal

- Base 16: on commence par 0x (ou 0X) suivi de chiffres hexadécimaux;

exemple: 0xFFF qui vaut 4095 en décimal

Par défaut, le type des constantes entières est le type int. On peut aussi spécifier explicitement le type en ajoutant l'un des suffixes L ou l (pour le type long), LL or ll (pour le type long long). On peut aussi ajouter le suffixe U ou u (pour unsigned). Des exemples sont fournis dans le tableau suivant :

Numération	Type	Exemple
Décimale	int long unsigned unsigned long	45 12L 27U 10UL
Octale	int	0775
Hexadécimale	int	0XFFF

Le type « nombre réel ou nombre en virgule flottante »

On distingue trois types de nombre en virgule flottante:

- float codé sur 32 bits
- double codé sur 64 bits
- et long double codé sur 80 bits

Exemple
double Pi=3,14159

Un nombre en virgule flottanteN peut représenter sous la forme suivante:
$N=s*M*(b)^{e}$
Où s : signe (1 pour un nombre négatif et 0 sinon)
M : mantisse
b : base
e : exposant

Format d'un nombre réel de type float

Chapitre 3:Les opérateurs

Opérateurs arithmétiques

- Addition (+)
- Soustraction (-)
- Division (/)
- Multiplication (*)
- Modulo ou le reste de division entière d'un entier par un autre (%)

Exemples :

a+b (addition de a et b)
a-b (soustraction de a et b)
a/b (quotient de a et b)
a*b (multiplication de a et b)
a%b (reste de la division euclidienne de a par b: 65%15 vaut 5

Opérateurs d'incrémentation (++) et de décrémentation

a^{++} : a est incrémenté c-à-d on ajoute 1 par l'opérande (a=a+1)
a^{--} : a est décrémenté c-à-d on retranche 1 par l'opérande (a=a-1)

L'opérateur préfixe ^{++}a (respectivement suffixe a^{++})incrémente a avant (respectivement) après de l'évaluer.
L'opérateur préfixe ^{--}a (respectivement suffixe a^{--})décrémente a avant (respectivement après de l'évaluer.

Exemples :
1 - Soit N = 4
- X= N^{++} ; signifie N=5 et X=4 (on affecte la valeur de N à X et après on incrément N)
- X= $^{++}$ N ; signifie N=5 et X=5 (on incrémente N puis on affecte N à X)

2 - Soit N=15
- Y= N^{++} +5; signifie N=16 et Y=20
- Y= $^{++}$ N -5; signifie N=16 et Y=11

Opérateurs d'affectation

On distingue deux types d'opérateurs d'affectation :

- l'opérateur d'affectation simple (=)
- L'opérateur d'affectation composé ou opérateur de raccourci (op=).

(op) est un opérateur arithmétique ou de manipulation de bits.

Les opérateurs d'affectation composés sont donc : += , -=, *=, /=, %=, ^=, &=, |=, >>=, <<=.

Exemples :

- x= y ; (assigne la valeur de y à x)
- x + = y est équivalent à x = x + y
- x - = y équivaut à x = x – y
- X*=y équivaut à x = x*y

Opérateurs relationnels

Toute comparaison est une expression de type int qui renvoie :

- La valeur 0 si le résultat de la comparaison est faux
- La valeur 1 si le résultat de la comparaison est vrai.

Le tableau suivant illustre les différents opérateurs relationnels

Opérateur	Traduction	Exemple	Résultat
<	Inférieur à	x < y	1 si x est inférieur à y
<=	Inférieur ou égal à	x <= y	1 si x est inférieur ou égal à y
>	Supérieur à	x > y	1 si x est supérieur à y
>=	Supérieur ou égal à	x >= y	1 si x est supérieur ou égal à y
==	Égalité	x == y	1 si x est égal à y
!=	Inégalité	x != y	1 si x est différent de y

Remarque :
Ne confond pas = opérateur d'affectation et == opérateur relationnel.

Opérateurs logiques

Les opérateurs logiques sont :

- && : et logique
- ||: ou logique
- ! : non logique

La table de vérité des opérateurs logiques est représentée dans le tableau en-dessous:

x	y	x && y	x || y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

1: signifie vraie
0: signifie faux
x && y est vrai ou vaut 1 si x et y sont toutes vraies
x || y vaut 1 si l'un au moins est vrai ou 1

x	! x
0	1
1	0

!x vaut 1 si x est 0

Opérateurs bit à bit

Les opérateurs bits à bits sont illustrés dans le tableau suivant :

Opérateur	Signification	Exemple	Résultat (pour chaque position de bit)
&	ET bit à bit	x & y	1 si les bits de x et y valent 1
|	OU bit à bit	x | y	1 si le bit de x et/ou de y vaut 1
~	NON bit à bit	~x	1 si le bit de x est 0
^	XOR bit à bit	x ^ y	1 si le bit de x ou de y vaut 1
<<	Décalage à gauche	x << y	Décale chaque bit de x de y position vers la gauche
>>	Décalage à droite	x >> y	Décale chaque bit de x de y position vers la droite

Exemple :

Opérande				Opération	Résultat	
x		y				
13	1011	10	1010	~x	4	0100
13	1011	10	1010	x & y	10	1010
13	1011	10	1010	x | y	13	1011
13	1011	10	1010	x ^ y	1	0001
13	1011	2	0010	x << y	44	101100
13	1011	2	0010	x >> y	2	0010

Chapitre 4: Les structures de contrôle

Instruction if … else

Syntaxe :

1. **if** (expression) Instruction1
2. **if** (expression) Instruction1 **else** Instruction2

La valeur d'expression est évaluée si elle est différente de 0, Instruction1 est exécutée; sinon Instruction2 est exécutée si elle existe. Instruction1 et Instruction2 peuvent être un bloc d'instruction.

Exemples :

1-
```
if(a>b){ printf(″a=%d est plus grand \n″, a);
        b=0;}
```

2-
```
if (x < y)  {printf (″y=%d » est max \n″, y) ; max = y ;}
else        {printf (″x=%d » est max \n″, y) ; max = x ;}
```

Instruction for

Syntaxe :

for (expression1 ; expression2 ; expression3)
Instruction (ou {bloc d'instruction})

expression1 : est évaluée une seule fois au début de la boucle, permet d'initialiser les paramètres de boucle.
expression2 : est évaluée et testée avant chaque passage dans la boucle (condition logique).
expression3 : est évaluée après chaque passage, permet l'incrémentation ou la décrémentation.
(Organigramme explication)

Exemple :

```
int i, Max=5 ;
for (i=0; i < Max ; i++)
{
    printf("Valeur de i : %d\n",i) ;
}
```

Instruction while

Syntaxe :

while (expression) Instruction

Il s'agit d'une boucle tant que expression est vraie, Instruction est exécutée.

Exemple :

```
#include<stdio.h>
#define Max 5
int i=0;
while (i < Max) {
   printf("Valeur de i : %d\n",i);
   i++;
}
```

Instruction do … while

Syntaxe :

do instruction **while** (expression);

L'instruction ou le bloc d'instruction est toujours exécutée au moins une fois. L'exécution des instructions est toujours répétée tant que l'expression testée est vraie.

Exemple :

```
#include <stdio.h>
#define MAX 5
int main() {
        int i=MAX;
        do {
                printf("Valeur de i : %d\n",i);
                i++;
        } while (i < MAX);
        return 0;
        }
```

Instruction switch

Syntaxe :

switch (expression) {
case constante$_1$: liste_d'instructions$_1$ **break** ;
….
case constante$_n$: liste_d'instructions$_n$ **break** ;
default : liste_d'instructions;
}

expression est évaluée, puis le résultat est comparé avec constante$_1$, constante$_2$ etc...
A la première constante$_i$ dont la valeur est égale à expression, la (ou les) liste(s) d'instructions correspondante(s) est exécutée jusqu'à la rencontre d'une instruction break. La rencontre d'un break termine l'instruction switch.

Instruction switch

Exemple:

```
#include <stdio.h>
main ( )
        { int n ;
        printf (”donner un entier : ”) ;
        scanf (”%d”, &n) ;
        while(getchar () != ‘\n’) ;
        switch(n)
                { case 0 : printf (”nul \n”) ; break ;
                  case 1 : printf (”un \n ”) ; break ;
                  case 2 : printf (”deux \n”) ; break ;
                  case 3 : printf (”trois \n”) ; break ;
                  case 4 : printf (”quatre \n”) ; break ;
                  default : printf (”grand \n”);
                }
        }
```

Instruction goto

Syntaxe :

goto etiquette ;

La directive **goto** permet de brancher directement à n'importe quel endroit de la fonction courante identifiée par une étiquette. Une étiquette est un identificateur suivi du signe ":".

Exemple :

```
for ( ... )
if ( erreur )
goto TRAITEMENT_ERREUR;
...
TRAITEMENT_ERREUR:
    printf("Erreur: ....");
```

Chapitre 5:Principales fonctions d'entrées-sorties standard

La fonction getchar

Syntaxe:

```
Var=getchar () ;
```

La fonction **getchar** permet la récupération d'un seul caractère à partir du clavier

Exemple :

```
#include <stdio.h>
int main()
{
        char c;
        printf("Entrer un caractère:");
        c = getchar();
        printf("Le caractère entré est %c\n",c);
        return 0;
}
```

La fonction putchar

Syntaxe:

putchar(var) ;

La fonction **putchar** permet l'affichage d'un seul caractère sur l'écran. Elle constitue la fonction complémentaire de **getchar**

Exemple :

```
#include <stdio.h>
int main()
{
        char c;
        printf("Entrer un caractère:");
        c = getchar();
        putchar (c );
        return 0;
}
```

La fonction puts

Syntaxe:

puts(ch) ;

Cette fonction affiche la chaîne de caractères ch.

Exemple :

```
#include <stdio.h>
int main() {
        char * toto = " Bonjour tout le monde!";
        puts(toto);
        return 0;
}
```

La fonction d'écriture à l'écran formatée printf

Syntaxe:

printf("chaîne de contrôle", variable$_1$, . . . , variable$_n$);

La chaîne de contrôle est composée du texte à afficher et les spécifications de format correspondant à chaque expression de la liste. Les spécifications de format ont pour but d'annoncer le format des données à visualiser. Elles sont introduites par le caractère %.

Exemple :

```
printf("%d\n",14);
printf("la valeur de a est %d\n",-14);
printf("|%+d|\n",14);
```

Les différents formats de la fonction printf

Le tableau suivant nous montre les différents formats de la fonction printf.

format		Écriture
%d	int	décimale signée
%ld	long int	décimale signée
%+d	int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long	décimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%o	unsigned int	octale non signée
%lo	unsigned long	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned int	hexadécimale non signée
%c	unsigned char	Caractère
%s	char*	chaîne de caractère

La fonction de saisie scanf

Syntaxe:

scanf("chaîne de contrôle", arg1, . . . , argn);

- La fonction scanf permet de récupérer les données saisies au clavier, dans le format spécifié.
- La chaîne de contrôle indique le format dans lequel les données lues sont converties. Elle ne contient pas \n
- arg1, . . . , argn : adressse de variables (variables précédées de &).

Exemple :

```
#include <stdio.h>
int main()
{
        int i;
        printf("entrez un entier  i =\n ");
        scanf("%d",&i);
        printf("i = %d\n",i);
        return 0;
}
```

Chapitre 6 : Les fonctions

- Une fonction est une découpage d'un programme (sous- programme).
- Un programme contient au moins une fonction *main* (programme principale)

et des fonctions dans lesquelles des instructions sont déportées.

- L'utilisation des fonctions permet de:
  - partager et réutiliser le code;
  - faciliter la lecture de grand programme;
  - Éviter les parties répétitives.

- En C, il n'y a que des fonctions (sous-programmes).
- Une fonction prend en entrée des **arguments**, et renvoie une **valeur en retour**.

Une fonction est composée de deux parties:

- d'une en-tête ou prototype de la fonction
- Le corps de la fonction

Format général d’une fonction :

```
Type_fonction nom_fonction(type1 arg1,…, typeNargN)
{
        [Déclaration des variables locales ]

        corps de la fonction
}
```

Exemple: fonction_somme

```
#include <stdio.h>
int fonction_somme(int a, int b)
{
        int tmp;
        tmp=a+b;
        return(tmp);
}
void main()
{
        int i=2;
        int j=9;
        int somme;
        somme=fonction_somme(i,j);
        printf("%d +%d = %d\n", i, j, somme );

}
```

Valeur en retour :

1. Une fonction renvoie une seule valeur du type de la fonction :

instruction ***return(valeur);***

- return = fin de la fonction,
- il est possible d'avoir plusieurs return dans une fonction,
- Return peut contenir une expression
- Si aucun type de donnée n'est précisé, le type *int* est pris par défaut.

Exemple :

```
int valeur_absolue (int n)
{
        if n>=0
                return (n);
        else
                return(-1*n);
}
```

2 . Une fonction peut ne rien renvoyer :

- Mot clé **void**
- Retourne rien : return();
- Typiquement : les fonctions d'affichage

Exemple :

```
void affiche_n_bonjour (int n)
{
        int i;
        for (i=0; i<n; i++)
        printf("bonjour");
        return;
}
```

EXERCICES

Chapitre 7 : Les tableaux

- Un tableau permet de stocker une liste de variables de même type.
- Stockage des variables en mémoire les uns à la suite des autres.

Tableau à une dimension

Déclaration d'un tableau à une dimension :

type nom_tableau [dimension_tableau]

- type: type des variables du tableau
- nom_tableau : nom du tableau qui est un identificateur
- dimension_tableau : nombre d'éléments du tableau, constante.

(= pas de variable)

Exemples :

```
#define N_max 5
int tab1[10]; // déclaration d'un tableau tab1 de 10 entiers de type int
float tab2[10]; // déclaration d'un tableau tab2 de 10 décimaux de type float
char tab3[N_max]; // déclaration d'un tableau tab3 de 5 caractère
                  //de type char
```

Initialisation de tableau :

On a deux façons pour initialiser un tableau:

- Soit à la déclaration
- Soit dans le code

Exemple 1: initialisation de tableaux

```
int tab1[10]={12,2,5,7,58,6,7,8,10,98};
int tab2[5]={1, 3, 4};
float tab3[2]={0.2, 6.7}
char tab4[6]={‘e’, ‘r’, ‘r’, ‘e’, ‘u’, ‘r’,’\0’};
int tab7[]={12, 5,8,9};
```

Exemple 2 : affichage des éléments d’un tableau

```
int tab[3]={1,2,3};
int i; // variable de la boucle
for (i=0;i<3;i++)
{
        printf (" %4d ",tab[i]);
}
```

Saisir les éléments du tableau

```
int main()
{
   int tab[10];
   int i; // variable du boucle
    printf("entrer les éléments du tableau\n");
   for (i=0;i<10;i++)
   {
      scanf ("%d", &tab[i]);
   }

   for(i=0;i<10;i++ )
   printf ("tab[%d]=%4d\n",i,tab[i]);

   return 0;
}
```

Tableau à plusieurs dimensions

Déclaration d'un tableau à plusieurs dimensions :

type nom_tableau [nombre_ligne][nombre_colonne];

- type: type des variables du tableau
- nom_tableau : nom du tableau qui est un identificateur
- nombre_ligne : nombre de lignes du tableau, constante.
- nombre_colonne : nombre de colonnes du tableau, constante.

Exemples :

Exemple1:

```
int tab1[10][10]; // déclaration d'un tableau tab1 de 10 lignes et 10 colonnes
                  //de type int
float tab2[10][10]; // déclaration d'un tableau tab2  10x10 décimaux
                    //de type float
```

Exemple 2:

```
int main()
{
   int  tab[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}
    int i,j;
    for(i=0;i<3;i++) /*boucle sur le ligne*/
    {
       for(j=0;j<4;j++) /* boucle sur le colonne*/
          {
             printf(" tab[%d][%d]=%d \n",i,j,tab[i][j]);
          }
    }

    return 0;
}
```

Initialisation d'un tableau à plusieurs dimensions :

exemples d'initialisation d'un tableau:

```
int tab1[2][3]={{1,2,3},{4,5,6}};
```

Ou bien `int tab1[2][3]={1,2,3,4,5,6};`

```
double tab2[3][3]={{4.2,5,9},{3.7,5,4},{4,3,9,12.5}};
```

Chapitre 8 : Les chaînes de caractères

- Pas de type particulier "chaîne" ou "string" en langage C.
- Une chaîne de caractères est traitée comme un tableau de caractères

à une dimension de type **char**.

- Lors de la déclaration (comme pour les tableaux de chiffres),

on doit indiquer l'espace à réserver.

- La représentation interne d'une chaîne de caractères est terminée

par le symbole '\0' (NUL), donc pour un texte de n caractères, il faudra réserver n+1 emplacements.

Déclaration :

```
char Nom_tableau [longueur];
```

Les chaînes de caractères constantes :

- Les chaînes de caractères sont indiquées entre guillemets.
- La chaîne de caractère vide s'écrit "" .
- Un caractère constant s'écrit toujours entre quotte (‘’).
- Dans les chaînes de caractères, on peut utiliser les séquences d'échappement (\n) comme un caractère constant.

Exemples:

char chaine[]="ceci est un exemple\n de texte qui \n s'écrira en 3 lignes\n" ;

char chaine[]={‘e’,’r’,’r’,’e’,’u’,’r’,’\0’};

Accès aux éléments d'une chaîne :

- L'accès aux éléments d'une chaîne se fait d'une manière que d'un tableau.

Exemples:

```
char chaine[7]={'e','r','r','e','u','r','\0'};

chaine[0]='e';
chaine[1]='r';
chaine[2]='r';
chaine[3]='e';
chaine[4]='u';
chaine[5]='r';
chaine[6]='\0';
```

Lecture et écriture des chaînes :

Exemple :

```
#include<stdio.h>

main()

{

        char nom[20], prenom[20];

       printf("donnez votre nom et prenom:\n");

        scanf("%s %s",nom, prenom);

        printf(" %s %s  \n", prenom, nom);

}
```

Remarque : Le nom de la chaîne est le représentant de l'adresse du premier caractère, il est donc inutile (et interdit!) de mettre &.

Tableaux des chaînes de caractères

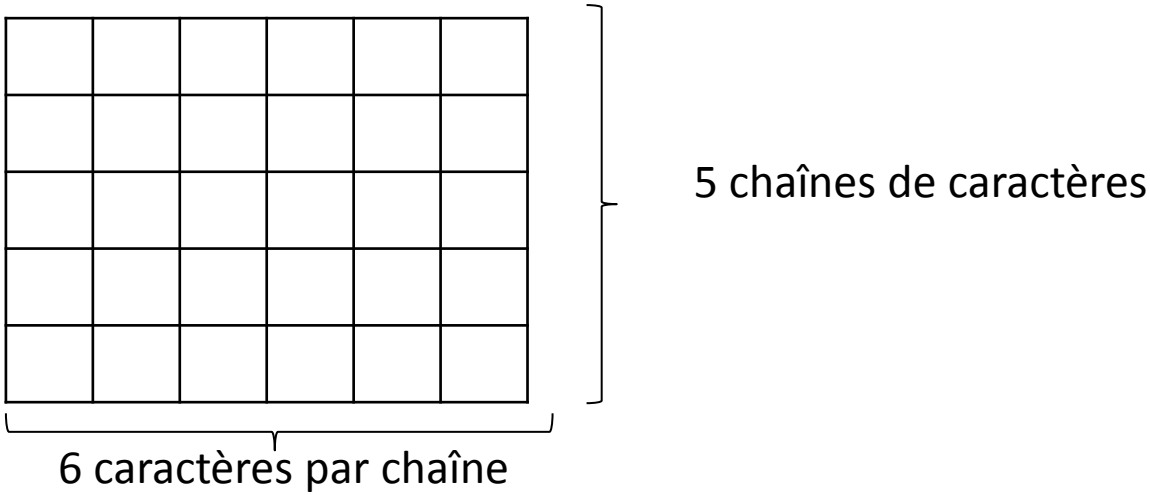
Déclaration :

Type nom[n_lignes][n_colonnes];

Un tableau qui réserve l'espace en mémoire pour n_lignes chaînes de caractères contenant n_colonnes caractères(n_colonnes significatifs)

Exemple 1:

```
char chaine[5][6]; //Un tableau qui réserve  l'espace en mémoire pour 5 chaînes
                   //de caractères contenant 6 caractères.
```



Les fonctions de string.h

La bibliothèque <string.h> fournit une multitude de fonctions pratiques pour le traitement ou la manipulation de chaînes de caractères.

Le tableau suivant représente ces fonctions:

Fonction	Explication
strlen(s)	Fournit la longueur de la chaîne sans compter le '\0' final
strcpy(s,t)	Copie t vers s
strcat(s,t)	Ajoute t à la fin de s
strcmp(s,t)	Compare s et t et fournit un résultat: - négatif : si s précède t - zéro: si s est égal à t - positif : si s suit t

Les symboles s et t peuvent être remplacés par:

- Une chaîne de caractères constante
- Le nom d'une variable déclarée comme tableau de caractère
- Un pointeur.

Chapitre 9 : Le pointeur

L'opérateur adresse & retourne l'adresse d'une variable en mémoire.

Exemple:

```
int i = 15;
printf("VOICI i: %d\n",i);
printf("VOICI SON ADRESSE EN HEXADECIMAL: %p\n",&i);
```

Définition:

Un pointeur est une variable qui peut contenir l'adresse d'une autre variable d'un type de données.

Si un pointeur P contient l'adresse d'une variable A, on dit que P pointe sur A.

Déclaration des pointeurs:

Une variable de type pointeur se déclare à l'aide de l'objet pointé précédé du symbole * (opérateur d'indirection).

Syntaxe :

type_objet_pointé *identificateur_pointeur;

Exemple:

```
char *pc; //pc est un pointeur qui peut recevoir des adresses
          //des variables de type char
int *pi;  //pi est un pointeur pointant sur un objet de type int
float *pf; //pf est un pointeur pointant sur un objet de type float
```

Les opérations élémentaires sur les pointeurs :

- L'opérateur « adresse de » : & pour obtenir l'adresse d'une variable.

Exemple :

```
Pa=&a; // on affecte à Pa l'adresse de a
Pt=&t; // on affecte à Pt l'adresse de t
```

- L'opérateur « contenu de » : *

*pointeur désigne le contenu de l'adresse référencée par le pointeur.

Exemple:

```
a et b deux variables quelconques
p: pointeur;

p=&a;// on affecte à p l'adresse de a
b=*p+1;//on affecte à b le contenu de l'adresse référencée
          //par p  ajouté par 1
*p=*p+10;
*p+=b;
```

Pointeur et tableau

En C, le nom d'un tableau représente l'adresse de son premier élément.

Exemple :

- Opérateur adresse de(&):

  tab=&tab[0];

  tab+1=&tab[1];

  ….

  tab+i=&tab[i];

- Opérateur contenu de (*):

  *tab=tab[0];

  *(tab+1)=tab[1];

  …

  *(tab+i)=tab[i];

Pointeur et chaîne de caractère

Exemple : exemple sur le tableau des chaînes de caractères

char * jour[7][9]={"lundi", "mardi", "mercredi", "jeudi, "vendredi",

"samedi", "dimanche"}

jour est un pointeur sur un tableau de chaînes de caractères qui réserve l'emplacement en mémoire de 7 chaînes de caractères contenant 9 caractères chacune.

Remarque :

Il est possible d'accéder aux différents chaînes de caractères d'un tableau en indiquant la ligne correspondante.

Exemple :

jour[1]=lundi;

jour[4]=jeudi;

Allocation dynamique de mémoire :

Un moyen de gérer la mémoire lors de l'exécution du programme.

Exemple:

```
char *nom[5];/* pour 5 pointeurs : 5xpoctets. Mais on ne peut savoir
                à l'avance le nombre d'octets à réserver pour les noms,
                puisqu'ils seront introduits que lors de l'exécution
                du programme.*/
```

Opérateur sizeof et fonction malloc :

Opérateur sizeof:

- sizeof (type); réservation de la mémoire des données d'un type .
- sizeof : retourne la grandeur de son argument.

sizeof(var) ; Fournit la grandeur de la variable "var"

sizeof(const) ; Fournit la grandeur de la variable "const"

sizeof(type) ; Fournit la grandeur d'un objet de type "type"

Exemple :

sizeof(int) → 2

sizeof(double) → 8

sizeof(4,5) → 8

int A[10]

sizeof(A) → 20

char B[5][10]

sizeof(B) → 50

Fonction malloc :

Une fonction qui permet d'allouer une place en mémoire à un pointeur.

Exemple:

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

Chapitre 10: Les structures et les énumérations

Structures :

Une structure est un objet composé de plusieurs champs qui sert à représenter un objet réel ou un concept.

Par exemple :

- Structure: voiture, champs: marque, modèle, couleur, année, …
- Structure: Etudiant, champs: nom, prénoms, âge, numéro matricule,….

Syntaxe:

```
struct  nom_de_structure
{
      type 1 nom_champ1;
      type 2 nom_champ2;
      type 3 nom_champ3;
       …
}variable;
```

Exemple:

```
struct date
{
        int jour;
        int mois;
        int annee;
};
```

```
struct personne
{
    char nom[20] ;
    char prénoms [20] ;
    int âge  ;
} ;
```

```
struct voiture
{
                char marque;
                char modèle;
                char couleur;
                int année;
};
```

```
struct Etudiant
{
                char nom[20];
                char prénoms[20];
                int note;

};
```

Accès aux champs de structure

```
#include <stdio.h>

typedef struct
 {
          int jour;
          int mois;
          int annee;
 } Date;
main()
 {
         Date d;
           d.jour=29;
           d.mois=10;
           d.annee=2015;
         printf("%d/%d/%d\n",d.jour,d.mois,d.annee);
       }
```

```
typedef struct
{
	char nom[20];
	char prenom[20];
	int age;
	float note;
}
fiche_etudiants;

void main()
{
	fiche_etudiants f;
	printf("SAISIE D'UNE FICHE \n");
	printf("NOM: ");gets(f.nom);
	printf("PRENOM: ");gets(f.prenom);
	printf("AGE: ");scanf("%d",&f.age);
	printf("NOTE: ");scanf("%f",&f.note);
	printf("\n\nLECTURE DE LA FICHE:\n");
	printf("NOM: %s PRENOM: %s AGE: %2d NOTE: %4.1f",f.nom,f.prenom,f.age,f.note);
}
```

Enumérations :

Un type énumération est un cas particulier de type entier et donc un type scalaire (ou simple).

Syntaxe :

```
enum nom_de_énumération {
        enumerateur1 ,
        enumerateur2 ,
        e t c .
        enumerateurN
} v a r i a b l e s ;
```

Exemple :

```
enum couleurs {
        rouge ;
        vert ;
        Bleu;
} rvb ;
```

Elle définit un type énumération nommé couleur et précise qu'il comporte trois valeurs possibles désignées par les identificateurs rouge, vert et bleu. Ces valeurs constituent les constantes du type couleur.

Les composants d'un enum sont des entiers numérotés de 0 à (n -1) sauf initialisation.

Chapitre 11: Fichiers

Opérations possibles avec les fichiers: Créer - Ouvrir - Fermer - Lire - Ecrire - Détruire - Renommer. La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H.

Création d'un fichier :

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

int main()
{   FILE *f;
   f=fopen("nomfic","w");
   if(!f)
   {
      printf("Le fichier n' est pas crée");
   }
   else
   {
     printf("Le fichier est crée");
   }
   fprintf(f,"bonjour!");

   return 0;
}
```

Ouverture et fermeture d'un fichier :

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

int main()
{   FILE *f;
    f=fopen("nomfic",« r");
    if(!f)
    {
        printf(« Impossible d'ouvrir le fichier");
    }
    else
    {
      fclose;
    }

    return 0;
}
```

Modes d’ouverture

"r" lecture seule, le fichier doit exister.

"w" écriture seule, destruction de l’ancienne s’il existe.

"a" écriture en fin de fichier, le fichier est créé s'il n'existe pas sinon son contenu sera étendu.

"r+" mise à jour (lecture écriture), le fichier doit exister. On ne peut enchaîner les deux opérations sans effectuer un fseek**.**

"w+" création pour mise à jour, le fichier est crée s'il n'existe pas ou s'il existe son contenu est perdu (de même que r+).

"a+" lecture et écriture d’un fichier existant (mise à jour), pas de création d’une nouvelle version si le fichier existe , le pointeur est positionné à la fin du fichier

Action	Instructions C
ouverture en écriture	f = fopen(nom, ″w″)
ouverture en lecture	f = fopen(nom,″r″)
fermeture	fclose(f)
fct fin fichier	feof(f)
Écriture d'une chaîne formatée Écriture d'un seul caractère Écriture d'une chaîne caractère	fprintf(f,″…″, adr) fputc fputs
Lecture	fscanf(f,″…″, adr) c = getc(f)