CHAPITRE 1

ELEMENTS DE LANGAGE C

Le corrigé des exercices et le listing de ces programmes se trouvent à la fin de chaque chapitre.

INTRODUCTION

Le langage C est un langage évolué et structuré, assez proche du langage machine destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel ...). Les compilateurs C possèdent les taux d'expansion les plus faibles de tous les langages évolués (rapport entre la quantité de codes machine générée par le compilateur et la quantité de codes machine générée par l'assembleur et ce pour une même application);

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

exemples: math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

On ne saurait développer un programme en C sans se munir de la documentation concernant ces bibliothèques.

Les compilateurs C sont remplacés petit à petit par des compilateurs C++.
Un programme écrit en C est en principe compris par un compilateur C++.
Le cours qui suit est un cours ce langage C écrit dans un contexte C++.

ETAPES PERMETTANT L'EDITION, LA MISE AU POINT, L'EXECUTION D'UN PROGRAMME

*1- **Edition du programme source***, à l'aide d'un éditeur (traitement de textes). Le nom du fichier contient l'extension .C, exemple: EXI_1.C (menu « edit »).

*2- **Compilation du programme source*****,** c'est à dire création des codes machine destinés au microprocesseur utilisé. Le compilateur indique les erreurs de syntaxe mais ignore les fonctions-bibliothèque appelées par le programme.
Le compilateur génère un fichier binaire, non listable, appelé fichier objet: EXI_1.OBJ (commande « compile »).

*3- **Editions de liens***: Le code machine des fonctions-bibliothèque est chargé, création d'un fichier binaire, non listable, appelé fichier executable: EXI_1.EXE .

*4- **Exécution du programme***.

Les compilateurs permettent en général de construire des programmes composés de plusieurs fichiers sources, d'ajouter à un programme des unités déjà compilées ...

Exercice I-1: Editer (EXI_1.C), compiler et exécuter le programme suivant:

```
#include <stdio.h>          /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
puts("BONJOUR");          /* utilisation d'une fonction-bibliotheque */
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits **en minuscules.**

On a introduit dans ce programme la notion d'interface homme/machine (IHM).
- L'utilisateur visualise une information sur l'écran,
- L'utilisateur, par une action sur le clavier, fournit une information au programme.

Modifier le programme comme ci-dessous, puis le tester :

```
#include <stdio.h>          /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
int a, b, somme ;  /* déclaration de 3 variables */
puts("BONJOUR");          /* utilisation d'une fonction-bibliotheque */
a = 10 ; /* affectation* /
b = 50 ;  /* affectation */
somme = (a + b)*2 ;  /* affectation et opérateurs   */
printf(« Voici le resultat : %d\n », somme) ;
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Dans ce programme, on introduit 3 nouveaux concepts :
- La notion de déclaration de variables : les variables sont les données que manipulera le programme lors de son exécution. Ces variables sont rangées dans la mémoire vive de l'ordinateur. Elles doivent être déclarées au début du programme.
- La notion d'affectation, symbolisée par le signe =.
- La notion d'opération.

LES DIFFERENTS TYPES DE VARIABLES

1- Les entiers

Le langage C distingue plusieurs types d'entiers:

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court non signé	2 octets: $0 \leq\leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq\leq n \leq 2^{7}-1$
unsigned char	caractère non signé	1 octet : $0 \leq n \leq 2^{8}$

Numération: En décimal les nombres s'écrivent tels que, précédés de 0x en hexadécimal.
exemple: 127 en décimal s'écrit 0x7f en hexadécimal.

Remarque: En langage C, le type **char** est un cas particulier du type entier:

un caractère est un entier de 8 bits

Exemples:

Les caractères alphanumériques s'écrivent entre ‘ ‘

Le **caractère** 'b' a pour valeur 98 (son code ASCII).
Le **caractère** 22 a pour valeur 22.
Le **caractère** 127 a pour valeur 127.
Le **caractère** 257 a pour valeur 1 (ce nombre s'écrit sur 9 bits, le bit de poids fort est perdu).

Quelques constantes caractères:

CARACTERE		VALEUR (code ASCII)	NOM ASCII
'\n'	interligne	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale	0x0b	VT
'\r'	retour charriot	0x0d	CR
'\f'	saut de page	0x0c	FF
'\\'	backslash	0x5c	\
'\"	cote	0x2c	'
'\"'	guillemets	0x22	"

Modifier ainsi le programme et le tester :

```
#include <stdio.h>          /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
int a, b, calcul ;  /* déclaration de 3 variables */
char u, v;
puts("BONJOUR");          /* utilisation d'une fonction-bibliotheque */
a = 10 ; /* affectation* /
b = 50 ;  /* affectation */
u = 65 ;
v = ‘A’ ;
```

```
calcul = (a + b)*2 ;  /* affectation et opérateurs   */
printf(« Voici le resultat : %d\n », calcul) ;
printf(« 1er affichage de u : %d\n »,u) ;
printf(« 2ème affichage de v : %c\n »,u) ;
 printf(« 1er affichage de u: %d\n »,v) ;
printf(« 2éme  affichage de v: %c\n »,v) ;
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

2- Les réels

Un réel est composé - d'un signe - d'une mantisse - d'un exposant
Un nombre de bits est réservé en mémoire pour chaque élément.

Le langage C distingue 2 types de réels:

TYPE	DESCRIPTION	TAILLE MEMOIRE
float	réel standard	4 octets
double	réel double précision	8 octets

LES INITIALISATIONS

Le langage C permet l'initialisation des variables dans la zône des déclarations:

char c; est équivalent à char c = 'A';
c = 'A';

int i; est équivalent à int i = 50;
i = 50;

Cette règle s'applique à tous les nombres, char, int, float ...

SORTIES DE NOMBRES OU DE TEXTE A L'ECRAN

LA FONCTION PRINTF

Ce n'est pas une instruction du langage C, mais une fonction de la bibliothèque stdio.h.

Exemple: affichage d'un texte:

printf("BONJOUR"); /* pas de retour à la ligne du curseur apres l'affichage, */
printf("BONJOUR\n"); /* affichage du texte, puis retour à la ligne du curseur. */

Exercice I-2: Tester le programme suivant et conclure.

```
#include <stdio.h>
#include <conio.h>
void main()
{
printf("BONJOUR ");
printf("IL FAIT BEAU\n");  /* equivalent à puts("BONJOUR");  */
printf("BONNES VACANCES");
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

La fonction printf exige l'utilisation de formats de sortie, avec la structure suivante:

printf("%format",nom_de_variable);

Exercice I-3: Affichage d'une variable de type **char:**

Tester le programme suivant et conclure.
Dans un deuxième temps, le modifier programme pour améliorer l'interface utilisateur.

```
#include <stdio.h>
#include <conio.h>
void main()
{
char c;
c =66;                  /* c est le caractere alphanumerique A */
printf("%d\n",c);       /* affichage du code ASCII en decimal */
                              /* et retour … à la ligne */
printf("%o\n",c);       /* affichage du code ASCII en base huit
                                 /* et retour … à la ligne */
printf("%x\n",c);       /* affichage du code ASCII en hexadecimal
                                 /* et retour … à la ligne */
printf("%c\n",c);       /* affichage du caractère */
```

```
                                /* et retour à la ligne */
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-4: Affichage multiple de structure:

printf("format1 format2 .... formatn",variable1,variable2, ...,variablen);

Tester le programe suivant et conclure:

```
#include <stdio.h>
#include <conio.h>
void main()
{
char c;
c ='A';/* c est le caractere alphanumerique A */
printf("decimal = %d  ASCII = %c\n",c,c);
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Formats de sortie pour les entiers:

%d affichage en décimal (entiers de type int),
%x affichage en hexadécimal (entiers de type int),
%u affichage en décimal (entiers de type unsigned int),

D'autres formats existent, consulter une documentation constructeur.

Exercice I-5:

a et b sont des entiers, a = -21430 b = 4782, calculer et afficher a+b, a-b, a*b, a/b, a%b en format décimal, et en soignant l'interface homme/machine.

a/b donne le quotient de la division, a%b donne le reste de la division.

Exercice I-6:

Que va-t-il se produire, à l'affichage, lors de l'exécution du programme suivant ?

```
#include <stdio.h>  /* ex I_6 */
#include <conio.h>
void main()
{
```

```
char a = 0x80;
unsigned char b = 0x80;
clrscr();
printf("a en decimal vaut: %d\n",a);
printf("b en decimal vaut: %d\n",b);
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-7:

En C standard, la taille des entiers est de 32 bits;
Que va-t-il se passer, à l'affichage, lors de l'exécution du programme suivant ?

```
#include <stdio.h>  /* ex I_7.C */
#include <conio.h>
void main()
{
int a = 12345000, b = 60000000, somme;
somme=a*b;
printf("a*b = %d\n",somme);
printf("a*b (en hexa) = %x\n",somme);
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Format de sortie pour les réels: %f

Exercice I-8:

a et b sont des réels, a = -21,43 b = 4,782, calculer et afficher a+b, a-b, a*b, a/b, en soignant l'interface homme/machine.

AUTRES FONCTIONS DE SORTIES

Affichage d'un caractère: La fonction **putchar** permet d'afficher un caractère:
c étant une variable de type **char**, l'écriture **putchar(c);** est équivalente à **printf("%c\n",c);**

Affichage d'un texte: La fonction **puts** permet d'afficher un texte:
l'écriture **puts("bonjour");** est équivalente à **printf("bonjour\n");**

Il vaut mieux utiliser puts et putchar si cela est possible, ces fonctions, non formatées, sont d'exécution plus rapide, et nécessitent moins de place en mémoire lors de leur chargement.

LES OPERATEURS

Opérateurs arithmétiques sur les réels: + - * / avec la hiérarchie habituelle.

Opérateurs arithmétiques sur les entiers: + - * / (quotient de la division) % (reste de la division) avec la hiérarchie habituelle.

Exemple particulier:
```
char c,d;
c = 'G';
d = c+'a'-'A';
```
Les caractères sont des entiers sur 8 bits, on peut donc effectuer des opérations. Sur cet exemple, on transforme la lettre majuscule G en la lettre minuscule g.

Opérateurs logiques sur les entiers:
& ET | OU ^ OU EXCLUSIF ~ COMPLEMENT A UN « DECALAGE A GAUCHE » DECALAGE A DROITE.

Exemples:
```
p = n « 3;    /* p est égale à n décalé de 3 bits à gauche */
p = n » 3;    /* p est égale à n décalé de 3 bits à droite */
```

L'opérateur **sizeof(type)** renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.

Exemple: `n = sizeof(char);  /* n vaut 1 */`

Exercice I-9: n est un entier (n = 0x1234567a), p est un entier (p = 4). Ecrire un programme qui met à 0 les p bits de poids faibles de n.

Exercice I-10: Quels nombres va renvoyer le programme suivant ?

```
#include <stdio.h>
#include <conio.h>
void main()
{
printf("TAILLE D'UN CARACTERE:%d\n",sizeof(char));
printf("TAILLE D'UN ENTIER:%d\n",sizeof(int));
printf("TAILLE D'UN REEL:%d\n",sizeof(float));
printf("TAILLE D'UN DOUBLE:%d\n",sizeof(double));
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

INCREMENTATION - DECREMENTATION

Le langage C autorise des écritures simplifiées pour l'incrémentation et la décrémentation de variables:

i = i+1; est équivalent à i++;

i = i-1; est équivalent à i--;

OPERATEURS COMBINES

Le langage C autorise des écritures simplifiées lorsqu'une même variable est utilisée de chaque côté du signe = d'une affectation. Ces écritures sont à éviter lorsque l'on débute l'étude du langage C car elles nuisent à la lisibilité du programme.

a = a+b; est équivalent à a+= b;
a = a-b; est équivalent à a-= b;
a = a & b; est équivalent à a&= b;

LES DECLARATIONS DE CONSTANTES

Le langage C autorise 2 méthodes pour définir des constantes.

1ere méthode: **déclaration** d'une variable, dont la valeur sera constante pour tout le programme:

Exemple:
```
void main()
{
const float PI = 3.14159;
float perimetre,rayon = 8.7;
perimetre =  2*rayon*PI;
....
}
```

Dans ce cas, le compilateur réserve de la place en mémoire (ici 4 octets), pour la variable pi, mais dont on ne peut changer la valeur.

2eme méthode: **définition d'un symbole** à l'aide de la directive de compilation **#define.**

Exemple:
```
#define PI = 3.14159;
void main()
{
float perimetre,rayon = 8.7;
perimetre =  2*rayon*PI;
....
}
```

Le compilateur ne réserve pas de place en mémoire. Les constantes déclarées par #define s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation.

LES CONVERSIONS DE TYPES

Le langage C permet d'effectuer des opérations de conversion de type: On utilise pour cela l'opérateur de "cast" ().

Exemple et exercice I-11:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=0x1234,j;
char d,e;
float r=89.67,s;
j = (int)r;
s = (float)i;
d = (char)i;
e = (char)r;
printf("Conversion float -> int: %5.2f -> %d\n",r,j);
printf("Conversion int -> float: %d -> %5.2f\n",i,s);
printf("Conversion int -> char: %x -> %x\n",i,d);
printf("Conversion float -> char: %5.2f -> %d\n",r,e);
printf("Pour sortir frapper une touche ");getch();
}
```

CORRIGE DES EXERCICES

Exercice I-5:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
a= -21430;
b= 4782;
printf("A + B = %d\n",a+b);
printf("A - B = %d\n",a-b);
printf("A x B = %d\n",a*b);
printf("A sur B = %d\n",a/b);
printf("A mod B = %d\n",a%b);
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-6:

a en décimal vaut -128 b en décimal vaut 128
Rques:
En C, le type **char** désigne un entier codé sur 8 bits.
-128 <= char <=+127 0<= unsigned char <= 255

Exercice I-8:

```
#include <stdio.h>    /* EXI_8*/
#include <conio.h>
void main()
{
float a,b;
a= -21.43;
b= 4.782;
printf("A + B = %f\n",a+b);
printf("A - B = %f\n",a-b);
printf("A x B = %f\n",a*b);
printf("A sur B = %f\n",a/b);
puts("Pour continuer frapper une touche...");
getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-9:

```
#include <stdio.h>
#include <conio.h>
main()
{
int n,p,masque;
clrscr();
n = 45;
p = 4;
printf("valeur de n avant modification:%x\n",n);
masque = ~0; /* que des 1 */
masque = masque « p;
n = n & masque;
printf("n modifié vaut:%x\n",n);
}
```

Exercice I-10:

En C standard: sizeof(char) vaut 1 sizeof(int) vaut 4 sizeof(float) vaut 4 sizeof(double) vaut 8.