C

MINIMUM

connaissances nécessaires à la programmation des microcontrôleurs en langage C

(une introduction au langage c A.N.S.I maj n°3)

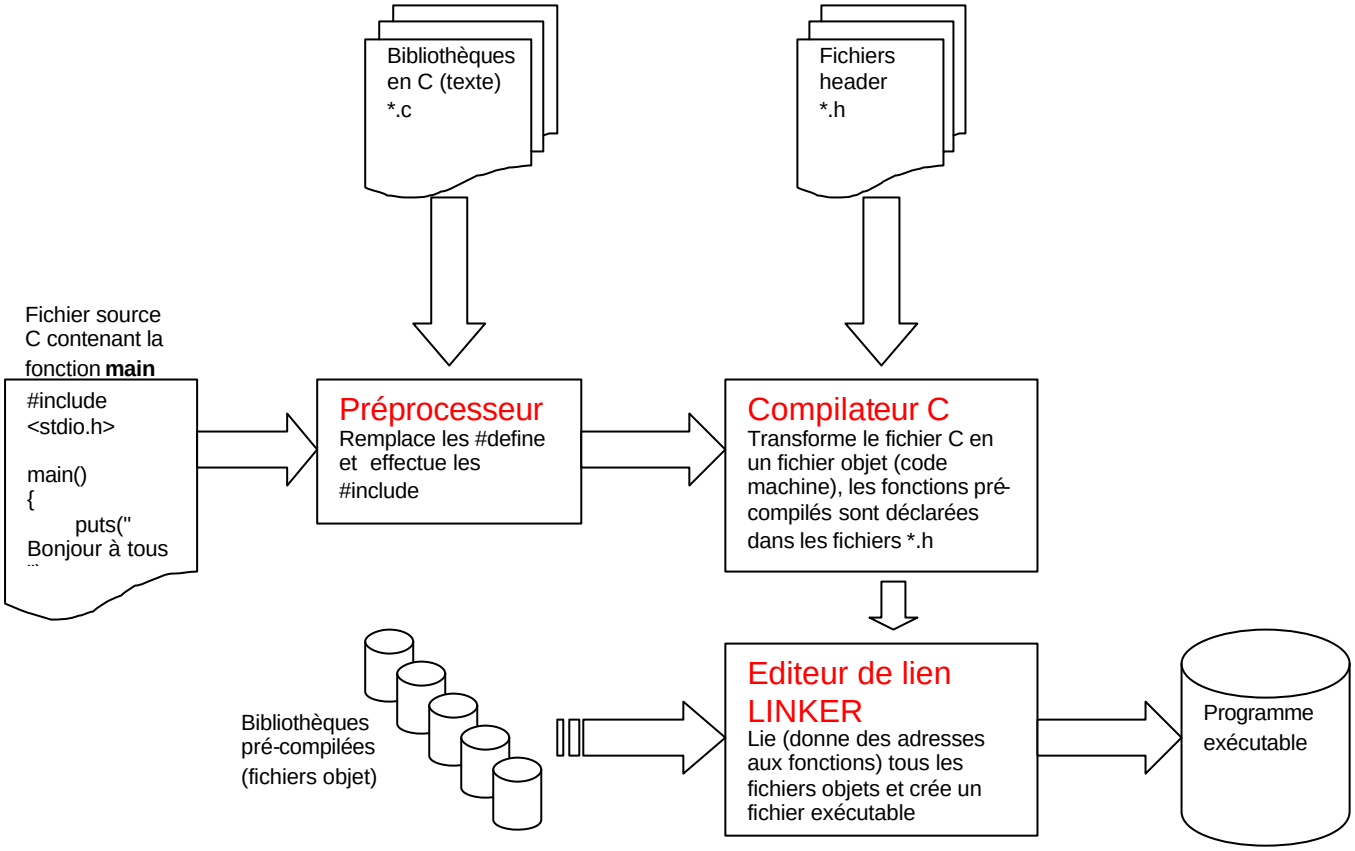
Christian Dupaty
Professeur de génie électrique
dupaty@unimeca.univ-mrs.fr

Compilateur C/C++ gratuits :
DOS : TURBO C/C++ V1.01 www.borland.fr
WIDOWS : Bloodshed DEV-C++ (GNU free software) www.bloodshed.net

Les lecteurs désirant approfondir leurs connaissances sont invités à consulter les cours sur l'algorithmique, le C, le C++ de P.Trau. http://www-ipst.u-strasbg.fr/pat/
Certains exemples de ce document proviennent de ce site

nombreux lies vers des sites traitant du C sur www.genelaix.fr.st

1. Organisation générale d'un compilateur C



2. LA SYNTAXE DU C : le premier programme

```
#define pi 3.14
#include <stdio.h>

float d,c;

int main()
{
d=2.0 ;
     c=pi*d ;
puts("bonjour à tous\n");
printf("la circonférence est %f m\n",c);
}
```

Remplacera pi par 3.14 (équivalence)

Header de la librairie contenant printf

Création de deux variables réelles

Programme principal

\n indique un passage à la ligne
%f indique l'affichage d'un nombre réel

affichera :
bonjour à tous
la circonférence est 6.28 m

LA SYNTAXE DU C :

```
/* convertisseur euro*/
#define TVA 19.6
#define taux 1.0/6.55957
```

Equivalences, elles sont remplacées par leurs valeurs par le pré processeur avant compilation

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

Librairies standards : *les fichiers « header » *.h contiennent en général des équivalences ou les prototypes des fonctions précomplilées ici :*
stdio pour printf, gets, puts
stdlib pour atof

```
char s[20];
float pht;
```

Variables globales :
char : octet
float réel

```
float calc_ttc(float prix)
{
float r;
      r=prix*(1.0+TVA/100.0)*taux;
      return(r);
}
```

Fonction (ou sous programme), en C il 'y a que des fonctions
Un paramètre réel en entrée, résultat sur un réel, du type y=sin(x)
r est une variable locale car déclarée dans la fonction, elle n'existe que lors de l'exécution de la fonction.

```
int main(void)
{
 do
  {
   puts("donnez le prix HT en francs");
   pht=atof(gets(s));
   printf("Prix HT en FRANCS : %f \n",pht);
   printf("Prix TTC en Euros : %f \n",calc_ttc(pht));
  }
 while (pht!=0.0);
 return 0;
}
```

main : fonction principale et point d'entrée du programme.
void indique qu'il n'y pas de paramètre d'entrée.
La fonction retourne un int=0, cela est obligatoire sur les systèmes MSDOS.
printf permet de formater les variables contrairement à **puts** (put string) qui se contente d'afficher une chaîne de caractère. \n indique un retour à la ligne
atof convertit une chaîne ASCII en un réel

3. VARIABLES, EQUIVALENCES ET CONSTANTES

Type	Longueur	Domaine de valeurs
char	8 bits	-128 à 127
unsigned char	8 bits	0 à 255
int	16 bits	-32768 à 32767
unsigned int	16 bits	0 à 65535
long	32 bits	-2,147,483,648 à 2,147,483,647
unsigned long	32 bits	0 à 4,294,967,295
float	32 bits	3.4 * (10**-38) à 3.4 * (10**+38)
double	64 bits	1.7 * (10**-308) à 1.7 * (10**+308)

Exemple de déclaration **char a,b,c ; /* trois caractères*/**

les données peuvent être regroupées en **tableaux** :

int table[100] ; /*tableau de 100 entiers*/
char tableau[]={10,0x1c,'A',55,4} ; /* tableau de 5 caractères*/
char *chaine= "bonjour" ; /*chaîne de 8 caractères (finie par 0)*/

le symbole ***** désigne un **pointeur** sur un type défini

char *p ; /* p est un pointeur sur des caractères*/

Equivalences : déclarées après la directive #define elles sont remplacées par leur valeur lors de la compilation

#define pi 3.14
#define fruit pomme !Attention il n'y a pas de **;** après une directive #define

Constantes : elles sont rangées dans la ROM (dans la RAM en lecture seule sur un PC) et ne sont donc pas modifiables.

const int i=16569, char c=0x4c ;

Variables: elles sont rangées dans la RAM soit à une adresse fixe (statique) soit dans une pile LIFO (dynamique)

char a,b=28,c='A' ; /* trois caractères dont 2 initialisés*/

auto est le contraire de **static** pour une variable locale. C'est une variable crée et détruite automatiquement (attribut par défaut).
near indique une adresse sur 16bits au contraire de **far** sur 21 bits
volatile indique une variable modifiable par l'environnement (un PORT par exemple)
CONST qui indique une contante (ROM). La distinction ROM/RAM n'est pas possible sur tous les systèmes (ex les PC) .

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

4. Boucles

For est utilisé lorsque l'on connaît à l'avance le nombre d'itérations d'une boucle.

Ex : **char i ;**

Condition de départ

Condition de fin

Evolution de la variable

```
For (i=0 ;i<100 ;i++)
      {
            printf("i = %d",i); /*affiche les i de 0 à 99*/
      }
```

(dans cet exemple les accolades sont superflues, il n'y a qu'une instruction dans la boucle)

```
char i=20 ;
For (;i<100 ;i++) printf("i = %d",i);      /* Pas de condition de départ*/
For( ;;) ;                                 /*une boucle sans fin non standard*/
```

While (expression) {instructions}, tant que l'expression est vraie (!=0) la boucle est effectuée**, la boucle peut ne jamais être effectuée**

```
i=0;
while (i<100)
      {
      printf("i = %d",i);      /*affiche les i de 0 à 99 */
      i++ ;
      }
```

Do {instructions }while (expression), comme while mais la boucle est effectuée au moins une fois

```
do
      {
      printf("i = %d",i); /*affiche les i de 0 à 99*/
      i++ ;
      }
While (i<100)
```

i aurait pu être remplacé par i++0

5. Branchements conditionnels :

if else

Une fonction qui est “vraie” si son paramètre est une voyelle

```
int calc(char c)
{
        if (c=='+') s=a+b; else
        if (c=='-') s=a-b; else
        if (c=='/') s=a/b; else
        if (c=='*') s=a*b;
        return(s);
}
```

Condition du test : ==, <,>,<=,>=, !=,&&,|| …

switch case

Le même fonction

```
int calc(char c)
{
switch (c )
   {
        case '+' : return (a+b);
        case '-' : return (a-b);
        case '*' : return (a*b);
        case '/' : return (a/b);
        default : return(0);
   }
}
```

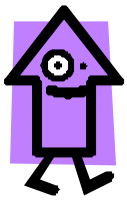
l’instruction **break** permet de sortir de la boucle en cours (for, while, do while, switch

l’instruction **continue** permet de sauter directement à l’itération suivante d’une boucle

```
for(i=0 ;i<100 ;i++) { if (i<50) continue else putchar(i);}
```

exit permet de quitter directement le programme (inutile sur micro contrôleur)

6. Les pointeurs



- Ce sont des **variables** particulières **qui contiennent l'adresse d'une variable**, elles ont toujours le même format, sur 68HC11 un pointeur est une valeur sur 16bits.
- Un pointeur est déclaré par une * précédée du type de donnée pointée
- Le signe & devant une donnée indique l'adresse de celle ci et sa valeur.
- **char *p ;** déclare un pointeur p sur un caractère
- **float *f ;** déclare une pointeur sur un réel.
- **char *fonction(void)** déclare une fonction qui retourne un pointeur sur un caractère
- **void (*fonction) (void)** déclare un pointeur sur une fonction
- **void (*fonction) (void) = 0x8000** crée un pointeur sur une fonction en 8000

exemples de manipulation de pointeurs

```
int    a=1,b=2,c ;        /*trois entiers dont deux initialisés*/
int    *p1,*p2 ;          /*deux pointeurs sur des entiers*/
p1=&a ;                   /*p1 contient l'adresse de la variable a*/
p2=p1 ;                   /*p2 contient maintenant aussi l'adresse de a*/
c=*p1 ;    /*c égale le contenue de l'adresse pointé par p1 donc c=a*/
p2=&b ;                   /*p2 pointe b*/
*p2=*p1                   /*la donnée à l'adresse pointé par p2 est placée
                          dans l'adresse pointé par p1, cela revient à donc
                          recopier a dans b*/
```

exemple d'utilisation des pointeurs : la fonction echange :

```
void echange(int i ,int j)
{
int k;
      k=i;
      i=j;
      j=k;
}
```

Lors de l'appelle par echange (x,y), les variables i,j,k sont localement crées dans la pile, i=x, j=y. i et j sont échangé mais pas x et y

La fonction echange qui fonctionne s'écrit comme cela :

```
void echange(int *i ,int *j)
{
int k
      k=*i ;
      *i=*j ;
      *j=k ;
}
```

I et j représente maintenant les adresses de x et y. k prend bien la valeur de x, i celle de j puis j celle de k. Les valeur x et y ont alors été échangées.

7. Tableaux

Un tableau est un regroupement dans une même variable de variables de même type

int chiffres[]={0,1,2,3,4,5,6,7,8,9} /* un tableau de 10 entiers*/

chiffre[0]=0, et chiffre[3]=3

Le premier indice d'un tableau est 0

int TAB[20]={1,12,13} /* les 17 autres sont initialisés à 0*/

TAB correspond à l'adresse de début du tableau, donc

- **TAB représente &TAB[0]**
- **TAB[0] représente *TAB**

TAB+1 pointera la donnée suivante et non l'adresse suivante

TAB+i = &TAB[i]

Un tableau peut avoir n dimensions

char TAB[2][3]={{1,2,3},{4,5,6}} représente une matrice 2x3 initialisée,

1	2	3
4	5	6

TAB[1][1]=5

Les chaînes de caractères sont des tableaux de caractères finissant par 0, *une chaîne est entourée de " et est automatiquement terminée par \0*

char message[]= "bonjour"; est la même chose que

char message[]={'b','o','n','j','o','u','r',\0} ;

on peut utiliser un pointeur pour traiter les chaînes de caractères

char *p= " bonjour " ;

while (*p !=0) putchar(*p++) ; /*équivalent à puts*/

Conversions de types : CAST

Lors d'un calcul les char sont automatiquement transformés en int.

Si nécessaire les transformations de type s'effectuent dans l'ordre

char -> int -> long -> float -> double

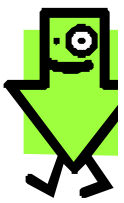
signed -> unsigned

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

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

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

Initialisation d'un pointeur à une adresse absolue



#define PORTA *(unsigned char *)(0x1000) ex: var=PORTA

la valeur 0x1000 est transformée en un pointeur sur un char. PORTA est équivalent au contenu de ce pointeur, donc au contenu de l'adresse 0x1000

8. Utilisation des pointeurs

(d'après « Le langage C » Kernighan et Ritchie Masson)

Soit deux chaînes de caractères : char s [ ],t[ ] ;
La fonction strcpy(s,t) recopie la chaîne s dans la chaîne t

```
void strcpy(char *s, char *t)
{
int i;
i=0;
do
    {
        s[i] =t[i]
        i++;
    }
    while (s[i-1] != '\0');
}
```

l'utilisation de pointeurs simplifie l'écriture

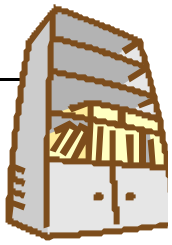
```
void strcpy(char *s, char *t)
{
while((*s=*t) != '\0')
    {
        s++ ;
        t++ ;
    }
}
```

on préfèrera écrire

```
void strcpy(char *s, char *t)
{
while((*s++=*t++) != '\0') ;
}
```

La proposition de la boucle tant que étant fausse si égale à zéro on peut écrire :

```
void strcpy(char *s, char *t)
{
  while (*s++=*t++)  ;
}
```



9. Bibliothèque standard stdio.h (très utilisée)

- **puts(chaîne)** ; affiche une chaîne de caractères

- **char *gets(chaîne)** ; saisie une chaîne de caractère au clavier finie par un RC et retourne un pointeur sue le premier caractère de cette chaîne

- **scanf(format, liste d'adresses)** permet de saisir les données au clavier
  Ex scanf("%d%d%f " ,&a,&b,&c) ;

attend la saisie de deux entiers puis d'un réel puis d'un RC. *Le passage d'argument par adresse est ici indispensable.*

- **printf**(format, liste de valeurs) affiche la liste de valeur dans un format choisi
  Ex char a=10 ; float b=3.1412
  printf(" décimal %d, hexa %x, reel %f " ,a,a,b) ;
  affichera : décimal 10, hexa A, reel 3,1412

Formats des types sur printf et scanf

- %c (char)
- %s (chaîne de caractères, jusqu'au \0)
- %d (int)
- %u (entier non signé)
- %x ou X (entier affiché en hexadécimal)
- %f (réel en virgule fixe)
- %p (pointeur)
- % (pour afficher le signe %).
- \ n nouvelle ligne
- \ t tabulation
- \ b backspace
- \ r retour chariot (même ligne)
- \ f form feed (nouvelle page)
- \ ' apostrophe
- \ \ antislash
- \ " double quote
- \ 0 nul

- **char getchar(void)** comme getch mais sur l'entrée standard

- **int putchar(char)** comme putch mais sur la sortie standard

Important : les fonctions puts, gets, printf, scanf etc.. utilisent pour acquérir ou envoyer un caractère getchar et putchar. Ce principe rend le C très universel, seules getchar et putchar diffèrent d'un système à l'autre. *(l'écran peut être un tube cathodique ou des cristaux liquides, le clavier peut être à 16 ou 120 touches …)*

Il en est de même pour les fonctions mathématiques qui reposent sur le principe des développements limités, seules + - / * doivent être définis par rapport au processeur cible.

Bibliothèques standards les plus utilisées

ctype.h	test pour détecter des types ex: isdigit (chiffre) ou islower (minuscule)
limits.h	indique les limites des types
string.h	traitement des chaînes, copie, concatène, recherche de sous chaîne etc.
math.h	fonctions mathématiques
stdlib.h	conversion ascii vers nombre (atoi atof) génération d'un nombre aléatoire (rand, srand) allocation dynamique de mémoire (malloc, calloc), tri (qsort)
time.h	toutes les fonctions liée à l'heure et à la génération de nombre aléatoires

10. Structures

Une structure est un tableau dans lequel les variables peuvent être de types différents

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

struct identite { char nom[30] ;
            char prenom[30] ;
            int age ;
            unsigned int tel;
          } classe[10] ;
char i;

int main()
{    strcpy(classe[0].nom,"dupont") ;
     strcpy(classe[0].prenom,"pierre") ;
     classe[0].age=40 ;
     classe[0].tel=4458;

     strcpy(classe[1].nom,"durand") ;
     strcpy(classe[1].prenom,"paul") ;
     classe[1].age=30 ;
     classe[1].tel=4454;

printf("\n\nprenom \tnom \tage \ttel\n");
for(i=0;i<2;i++)    {
printf("%s\t%s\t%d\t%d\n",classe[i].prenom,classe[i].nom,class
e[i].age,classe[i].tel);
                }
  return 0;
}
```

Création d'un type de structure identite composée de données de types différents
La variable classe crée est de ce type

Accès aux données de la structure

11. Champs de bits

On peut créer une structure « champ de bits ». Le premier élément est le bit 0. Le nom de l'élément est suivi du nombre de bits utilisés.

```
struct {
        unsigned RB0:1;
        unsigned RB1:1;
        unsigned RB2:1;
        unsigned RB3:1;
        unsigned GROUPE:3;
        unsigned RB7:1;
      } PORTBbits ;
```

```
char c ;

c=PORTBbits.RB2 ;
PORTBbits.RB3=1 ;
```

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RB7	GROUPE			RB3	RB2	RB1	RB0

12. Union

Dans une UNION les champs partagent les mêmes adresses.

```
volatile near union  {
    struct {
        unsigned RE0:1;
        unsigned RE1:1;
        unsigned RE2:1;
        unsigned RE3:1;
        unsigned RE4:1;
        unsigned RE5:1;
        unsigned RE6:1;
        unsigned RE7:1;
     } ;
    struct {
        unsigned ALE:1;
        unsigned OE:1;
        unsigned WRL:1;
        unsigned WRH:1;
        unsigned :3;
        unsigned CCP2:1;
           } ;
    struct {
        unsigned AN5:1;
     } ;
} PORTEbits ;
```

```
PORTEbits.RE0
PORTEbits.ALE
PORTEbits.AN5
Partagent le même bit physique
```

13. La récursivité en langage C

L'agorithme de tri QuickSort a été inventé par C.A.R Hoare en 1960. Il consiste à trier une partie d'un tableau, délimitée par les indices gauche et droite. On choisit une valeur quelconque de ce sous tableau. On recherche ensuite la position définitive de cette valeur, en plaçant toutes les valeurs inférieurs d'un coté et toutes les valeurs supérieurs de l'autre sans les classer. On appelle trie ensuite de manière récursive le coté des plus petits et celui des plus grands et cela jusqu'à ce que les cotés traités soient réduits à un seul élément.

```
#include <stdio.h>
typedef int type_vt;     /* types variables*/
typedef type_vt *type_pt;/* pointeurs de variables*/
type_vt table[]=
{10,5,12,4,8,25,57,4,15,18,14,38,50,44,8,77,18,26,56,
111};
char d;

void tri_rapide(type_pt tab,int gauche,int droite)
 {
  int g,d;
  type_vt tampon,val;
  if(droite<=gauche)return;
  val=tab[droite];  /*choix du pivot: arbitraire*/
  g=gauche-1;
  d=droite;
  do
   {while(tab[++g]<val);
    while(tab[--d]>val);
  if(g<d){tampon=tab[g];tab[g]=tab[d];tab[d]=tampon;}
   }
  while(g<d);
 tampon=tab[g];tab[g]=tab[droite];tab[droite]=tampon;
 tri_rapide(tab,gauche,g-1);
 tri_rapide(tab,g+1,droite);
 }

int main()
{for(d=0;d<20;d++) printf("%d ",table[d]);puts("\n");
 tri_rapide(table,0,19);
 for(d=0;d<20;d++) printf("%d ",table[d]);puts("\n");
 d=getchar();
 return 0;
}
```

14. Les réels

Représentation en virgule fixe

On affecte à la partie entière et à la partie décimale un certain nombre de bits. Le poids des bits est positif pour la partie entière et négatif pour la partie décimale

L'erreur absolue sur un réel représenté en virgule fixe est toujours inférieure à 2^{-m}

Exemple : pour représenter le réel 5,635 sur 8 bits (4 pour la partie entière et 4 pour la partie décimale) max 15,9375. on obtient :

4+1+0.5+0.125 = 5.625 nombre le plus proche

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
0	1	0	1	1	0	1	0

Soit 0x5A

Un bit supplémentaire est nécessaire pour indiquer le signe (+/-)

2^n	……
2^7	128
2^6	64
2^5	32
2^4	16
2^3	8
2^2	4
2^1	2
2^0	1
2^{-1}	0,5
2^{-2}	0,25
2^{-3}	0,125
2^{-4}	0,0625
2^{-5}	0,03125
2^{-6}	0,015625
2^{-7}	0,0078125
2^{-m}	……

Représentation en virgule flottante

$$r = Sm.M.10^{Se.E}$$

r : réel à coder
Sm : signe de la matrice (0 = positif, 1 = négatif)
M : matrice
Se : signe de l'exposant
E : exposant

Un nombre réel codé en virgule flottante a cette aspect :

Sm	Se	En	…	E0	Mm	…	M0

16, 32, 64 ou128 bits suivant les processeurs et les compilateurs

Nb bits	Exposant N	Mantisse M
16	4	10
32	8	22
64	14	48

La mantisse représente les chiffres significatifs d'un réel inférieur à 0 codé en 2^{-n}
Par exemple 245,36 a une mantisse égale à +24536 et un exposant égale à +3 :
$245,36=0.24536.10^3$

Exemple de codage en virgule flottante :
$-5,635 = -0,5635.10^1$ et $2^{-1}+2^{-4}+2^{-10}=0.5634765$

Sm	Se	E3	E2	E1	E0	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1

1 0.5634765

15. Les opérateurs

Fonctions	O	Description	Exemples
Identificateurs	**()**	Appel de fonction	
	[]	Indice de tableau	tableau[3]=5;
opérateurs unaires	**!**	Négation logique (NOT)	b=!a; (si a>0 => b=0, si a=0 =>b=1)
	~	Complément binaire bit à bit	b=~a
	-	Moins unaire	b=-a;
	+	Plus unaire	b=+a;
	++	Préincrément ou postincrément	b=a++; (b=a puis a=a+1)
	--	Prédécrément ou postdécrément	b=a--; (b=a puis a=a-1)
	&	Adresse de	b=&a; (b égale l'adresse de a)
	*****	Indirection (adressage indexé)	b=*a; (b=contenu de l'adresse de a)
opérateurs binaires	*****	Multiplication	c=a*b;
	/	Division	c=a/b;
	+	Plus binaire	c=a+b;
	-	Moins binaire	c=a-b;
	<<	Décalage à gauche	c=a<<b; (a est décalé b fois à gauche)
	>>	Décalage à droite	c=a>>b; (a est décalé b fois à droite)
	&	ET entre bits	c= a & b; (ET logique bit à bit)
	^	OU exclusif entre bits	c= a ^b;
	|	OU entre bits	c= a|b;
Tests	**<**	Strictement inférieur	if a < b
	<=	Inférieur ou égal	if a >= b
	>	Strictement supérieur	if a > b
	>=	Supérieur ou égal	if a >= b
	==	Egal	if a ==b (si a est égale à b)
	!=	Différent	if a != b
	&&	ET logique	if ((a=5) && (b=2))
	||	OU logique	if ((a=5) ||(b=2))
	?:	Condition	z=(a>b)?a:b (Si a>b a z=a sinon z=b)
Affectation	**=**	Affectation simple	a=b; (a prend la valeur de b)
Auto-affectations	***=**	Affectation produit	a*=2 (a=a*2)
	/=	Affectation quotient	a/=2 (a= a/2)
	%=	Affectation reste	a%=2 (a= le reste de a/2)
	+=	Affectation somme	a+=2 (a=a+2)
	-=	Affectation différence	a-=2 (a=a-2)
	&=	Affectation ET entre bits	a&=5 (a=a&5)
	^=	Affectation OU EX entre bits	a^=5 (a=a^5)
	|=	Affectation OU entre bits	a|==5 (a=a|=5)
	<<=	Affectation décalage gauche	a<<=5 (a=a<<5)
	>>=	Affectation décalage droite	a>>=5 (a=a>>5)

conditionnel ? :
C'est un (le seul) opérateur ternaire. L'expression a?b:c vaut la valeur de b si a est vrai (entier, différent de 0), et c si a est faux. **Exemple** : max=a>b?a:b

Dans une expression logique le second élément n'est évalué que si nécessaire : ex if ((a==0) || (b++==0)) …. B sera incrémenté si a !=0

16. EXEMPLES d'utilisation des opérateurs

/* OPÉRAT1.C : Programme d'exemple des Opérateurs de base et tests logiques pour les types de base */

```
#include <stdio.h>
char a, b, c;
int i1, i2, i3, i4;
long l1, l2;
float f1, f2, f3;
double d1;

main()
{
  i1 = i2 = i3 = 11;          /* voila des exemples d'affectations multiples */
  a = b = c = 40;
  f1 = f2 = f3 = 12.987;

  /* Opérations arithmétiques de base */
  i3 = -i2;                           /* donne i3 = -11 */
  i3 = i1 + i2;                 /* donne i3 = 22 */
  i3 = i1 - i2;                 /* donne i3 = 0 */
  i3 = i1 * i2;                 /* donne i3 = 121 */
  i3 = i1 / i2;                 /* donne i3 = 1 */
  i3 = 15+(i1 = 4*i2);                /* donne i1= 44 puis i3 = 59 */
  i4 = i3 / i2;                 /* donne i4 = 5 */
  i4 = i3 % i2;                 /* % = modulo donc i4 = 8 */
  i4 = i3 >> 2;           /* i4 = i3 apr,s 2 décaléges … droite (*4) */
  i4 = i3 << 3;           /* i4 = i3 apr,s 3 décaléges … gauche (/8) */

  /* Opérations arithmétiques de niveau 2 : automodifications */
  a = a + 1;                          /* incrémentation de a */
  a += 1;                             /* idem */
  a++;                                /* idem APRES utilisation de a */
  ++a;                                /* idem AVANT utilisation de a */
  a = a - 1 ;                         /* décrémentation de a */
  a -= 1;                             /* idem */
  a = 40;
  b = a--;                            /* b = 40 puis a = 39 */
  b = --a;                            /* a = 38 puis b = 38 */
  a += 10;                            /* a+10, a = 48 */
  b -= 5;                             /* b-5, b = 33 */
  f1 *= 1.5;                          /* f1 * 1.5, f1 = 8.658 */
  f2 /= 2.2;                          /* f2 / 2.2, f2 = 5.9032 */
  a <<= 1;                            /* a décalé 1 fois à gauche a = 24 */
  b >>= 2;                            /* b décalé 2 fois à droite b = 132 */

  /* Opérateurs Logiques de niveau 1 */
  b = 15; c = 12;
  a = b & c;                          /* a = b ET c donc a = 10 */
  a = b | c;                          /* a = b OU c donc a = 17 */
  a = b ^ c;                          /* a = b OU EXCLUSIF c donc a = 7 */
  a = ~b;                             /* a = NOT b donc a = $EF */

  /* Opérations de niveau 3 : Affectation conditionnelle */
  /* syntaxe générale : (Test) ? action_si_oui : action_si_non; */
  i1 = (i2 > 3) ? 2 : 10;   /* Cette instruction doit se lire:
                               si i2 est supérieur à 3 alors i1 devient 1
                               sinon, i1 devient 10 */
       c = (a > b) ? a : b;           /* c est le maxi d'entre a et b */
  c = (a > b) ? b : a;          /* c est le mini d'entre a et b */
```

```
   c = (a >= 0) ? a : -a;   /*  c est la valeur absolue de a */

   /* Opérateurs divers */
   a = sizeof (f1);                  /* sizeof retourne la taille en octet de */
   b = sizeof (l1);                  /* l'expression paramètre */
   f2 = int(15) / int(4);     /* conversion de type : CAST. En absence des */
   f2 = int (15/4);           /* commandes de cast, f2 vaudrait 3.75, ici 3 */

   /* Comparaisons logiques de base */
   if (i1 == i2) i= -13;            /* puisque i1 = i2, i3 deviendra -13 */
   if (i1 > i3) a = 'A';             /* donne 'A' (65) … la variable a */
   if (!(i1 > i3)) a = 'B';          /* puisque i1 >i3, a ne change pas */
   if (b <= c) f1 = 0.0;             /* puisque b = c alors f1 devient 0 */
   if (f1 != f2) f3 = i3/2;          /* comme f1 <> f2, f3 devient 6.5 */

   /* Comparaisons logiques de niveau 2 */
   /* bas,es sur le fait qu'une valeur vraie est diff,rente de 0 et qu'une
   valeur fausse est nulle */
   if (i1 = (f1 != f2)) i3 = 111;      /* comme f1 <> f2, le résultat du test
                                          est vrai, le résultat logique du test
                                          vrai est affecté à i1 et i3 change */
   if (i1) i3 = 222;                   /* i1 est vrai donc i3 change */
   if (i1 != 0) i3 = 333;            /* i1 vrai donc non nul donc i3=333 */
   if (i1 = i2) i3 = 444;            /* i2 recoit i1, cette valeur non nulle
                                          donc vraie  i3 = 222 */

   /* Comparaisons logiques niveau 3 : Tests multiples */
   i1 = i2 = i3 = 77;
   if ((i1 == i2) && (i1 == 77)) i3 = 33;    /* && = ET logique: i3 change */
   if ((i1 > i2) || (i3 > 12)) i3 = 22;     /* || = OU logique: i3 change */
   if (i1 && i2 && i3) i3 = 11;                  /* tous non nuls donc i3 change */
   if ((i1=1)&&(i2=2)&&(i3=3)) i3 = 44;          /* Attention, affectations des
                                                    variables donc i3 change */
   if ((i1==2)&&(i2=3)) i3 = 55;         /* i1 <> 2 donc i3 ne change pas */
}
```

17. Exemples de programmes en C

/* AFFLIGNE.C : Ce programme affiche des lignes avec des commandes de tabulations (\t) et de sauts de lignes (\n) */

```
#include <stdio.h>
main ()
{   printf ("Ceci est une ligne de texte suivi d'un saut de ligne\n");
   printf ("Cette deuxième ligne recoit -->");
   printf ("un texte attaché\n");
   printf ("Voici une ligne \ncoup,e \n\npar 3 sauts de lignes");
   printf ("\n\tEt une tabulation! \tet 2\tet 3\tet 4!\n");
}
```

/* CONTBRK.C : Ce programme présente les effets des instructions CONTINUE et BREAK */

```
#include <stdio.h>
main ()
{
        int index;
        printf ("\n\n\t présentation de l'interruption de boucle par BREAK \n");
        for (index = 4; index < 10; index ++)
        {
                if (index == 8) break;
                printf ("\nLa valeur du compteur est actuellement %d", index);
        }
        printf ("\n\n\t présentation de la suspension de boucle par CONTINUE \n");
        for (index = 4; index < 10; index ++)
        {
                if (index == 8) continue;
                printf ("\nLa valeur du compteur est actuellement %d", index);
        }
}
```

/* SWITCH.C : ce programme démontre l'usage de la commande d'aiguillage en C elle utilise les mots clefs SWITCH, CASE, BREAK */

```
#include <stdio.h>
main ()
{       int index;
        for (index = 3; index < 13; index++)
        {                               /* début de la boucle for */
                switch (index)          /* étude de la  valeur de index */
                {
                        case 3 : printf ("\n la valeur de l'index est 3 ");
                                        break;
                        case 4 : printf ("\n la valeur de l'index est 4 ");
                                        break;
                        case 5 :
                        case 6 :
                        case 7 :
                        case 8 : printf ("\n la valeur est comprise entre 5 et 8");
                                        break;
                        case 11 : printf ("\n à présent la valeur est 11 !");
                                        break;
                        default : printf ("\nles valeurs entre 8 et 10");
                                        printf (" et supérieures à 11 ne sont pas commentées");
                                        break;
                }                       /* fin du bloc switch */
        }               /* fin du bloc for */
}         /* fin de main() */
```

/* STRUCT.C : Programme d'exemple des structures de base DO ... WHILE
IF ... ELSE (imbriquées), WHILE, et FOR */

```
#include <stdio.h>
#include <stdlib.h>        /* pour l'accès aux fonctions rand, srand et time */
int cache, propose, trouve, compteur = 0;
char rep;
main()
{
  do
  {
    printf ("\n\nVous devez trouver en 10 coups un entier compris entre 0 et 50");
    srand (time(NULL));
    cache = rand() % 50;
    trouve = 0; rep = ' ';
    for (compteur = 1; compteur < 10; compteur += 1)
    {
          printf ("\n\tEntrer votre propostion : ");
          scanf ("%d", &propose) ;
          if (propose == cache)
          {
            printf ("\n\t\t BRAVO  -  C'est gagné en %d coups!", compteur);
            trouve = 1;
            compteur = 10;          /* ce qui permet de forcer la sortie du FOR */
          }
          else
            if (propose > cache)
              printf ("\n\tC'est trop grand");
            else
              printf ("\n\tC'est trop petit");
    }
    while (rep != 'O' && rep != 'N')
    {
          printf ("\n\nVoulez vous rejouer [O]ui/[N]on");
          scanf ("%c", &rep);
    }
  }
  while (rep == 'O');
  printf ("\n\n\tAu Revoir !!");
}
```

/* exemple de pilotage du port // d'un PC*/

```
#include <dos.h>              /* pour outportb et inportb */
#include <conio.h>            /* pour clrscr */
#include <stdio.h>            /* pour la fonction printf, a retirer */
char n =0x55;
void main ()
{   clrscr ();
          do
          {          gotoxy (10,10);
                     outportb(0x379,n) ;
                     printf ("Port entrée // vaut :%xb \n",inportb (0x379));
                     printf ("%xb est envoyé sur le port sortie // ",n));
          }
          while (1);
}
```

18. Exercices sur PC:

1. Un nom est saisi au clavier (ex : robert) puis l'ordinateur affiche " Bonjour robert " ,
   Utiliser *Scanf et printf*

2. Après avoir entré la longueur et la largeur le programme retourne le périmètre,
   Utiliser une fonction : float perimetre(float l,float L) ;

3. Réaliser une fonction void copiemem(*a,*b,long) qui recopie long octets de l'adresse a vers l'adresse b (a et b peuvent être des tableaux ou des chaînes de caractères)

4. Essayer le programme de test de QuickSort du cours, l'analyser, le faire fonctionner en pas à pas afin de faire apparaître la récursivité.

5. Pour la charge de C dans R (charge initiale nulle), après avoir entré R,C et la tension finale, le programme affiche vs pour 10 valeurs de t comprises entre 0s et 5tau
   Utiliser Math.h, for

6. Rechercher un nombre aléatoire entre 0 et 999, à chaque essai le programme indique " trop grand " ou " trop petit " et en cas de réussite le nombre d'essais,
   Utiliser If, do while, rand

7. Afficher les puissances de 2 jusqu'à 16000
   Utiliser for

8. Réaliser une calculatrice 4 opérations
   Utiliser case

9. Rechercher les nombres premiers
   Utiliser % ou div pour trouver le reste d'une divison

10. Jeux des allumettes
    Au départ on place sur le jeu N allumettes, on décide du nombre d'allumettes que l'on peu ôter à chaque tour (on doit ôter au moins une allumette et au maximum le nombre convenu), chaque joueur ôte à tour de rôle des allumettes, le perdant est celui qui prend la dernière.
    A) Réaliser un programme de jeux des allumettes pour deux joueurs humains, avec affichage du nombre d'allumette sur l'écran à chaque tour (avec le carctère I par exemple)
    B) Remplacer l'un des joueurs par l'ordinateur. (astuce lors de son tour l'ordinateur otera : **nbal-(((nbal-1)/(max+1))*(max+1)+1)** allumettes,
    avec **nbal** : nombre d'allumettes restant et **max** : nombre max d'allumettes ôtables

11. Pilotage du port paralèlle (uniquement sous win 95/98)
    Réaliser un clignotant sur le port //
    Réaliser un chenillard sur le port // (utiliser >> et <<)
    Utiliser outportb, inportb (port // en entrée en 0x378, en sortie en 0x379)

Annexe : Nombres premiers de 1 à 213

1: 1	55: 257	109: 599	163: 967	217: 1327	271: 1741
2: 3	56: 263	110: 601	164: 971	218: 1361	272: 1747
3: 5	57: 269	111: 607	165: 977	219: 1367	273: 1753
4: 7	58: 271	112: 613	166: 983	220: 1373	274: 1759
5: 11	59: 277	113: 617	167: 991	221: 1381	275: 1777
6: 13	60: 281	114: 619	168: 997	222: 1399	276: 1783
7: 17	61: 283	115: 631	169: 1009	223: 1409	277: 1787
8: 19	62: 293	116: 641	170: 1013	224: 1423	278: 1789
9: 23	63: 307	117: 643	171: 1019	225: 1427	279: 1801
10: 29	64: 311	118: 647	172: 1021	226: 1429	280: 1811
11: 31	65: 313	119: 653	173: 1031	227: 1433	281: 1823
12: 37	66: 317	120: 659	174: 1033	228: 1439	282: 1831
13: 41	67: 331	121: 661	175: 1039	229: 1447	283: 1847
14: 43	68: 337	122: 673	176: 1049	230: 1451	284: 1861
15: 47	69: 347	123: 677	177: 1051	231: 1453	285: 1867
16: 53	70: 349	124: 683	178: 1061	232: 1459	286: 1871
17: 59	71: 353	125: 691	179: 1063	233: 1471	287: 1873
18: 61	72: 359	126: 701	180: 1069	234: 1481	288: 1877
19: 67	73: 367	127: 709	181: 1087	235: 1483	289: 1879
20: 71	74: 373	128: 719	182: 1091	236: 1487	290: 1889
21: 73	75: 379	129: 727	183: 1093	237: 1489	291: 1901
22: 79	76: 383	130: 733	184: 1097	238: 1493	292: 1907
23: 83	77: 389	131: 739	185: 1103	239: 1499	293: 1913
24: 89	78: 397	132: 743	186: 1109	240: 1511	294: 1931
25: 97	79: 401	133: 751	187: 1117	241: 1523	295: 1933
26: 101	80: 409	134: 757	188: 1123	242: 1531	296: 1949
27: 103	81: 419	135: 761	189: 1129	243: 1543	297: 1951
28: 107	82: 421	136: 769	190: 1151	244: 1549	298: 1973
29: 109	83: 431	137: 773	191: 1153	245: 1553	299: 1979
30: 113	84: 433	138: 787	192: 1163	246: 1559	300: 1987
31: 127	85: 439	139: 797	193: 1171	247: 1567	301: 1993
32: 131	86: 443	140: 809	194: 1181	248: 1571	302: 1997
33: 137	87: 449	141: 811	195: 1187	249: 1579	303: 1999
34: 139	88: 457	142: 821	196: 1193	250: 1583	304: 2003
35: 149	89: 461	143: 823	197: 1201	251: 1597	305: 2011
36: 151	90: 463	144: 827	198: 1213	252: 1601	306: 2017
37: 157	91: 467	145: 829	199: 1217	253: 1607	307: 2027
38: 163	92: 479	146: 839	200: 1223	254: 1609	308: 2029
39: 167	93: 487	147: 853	201: 1229	255: 1613	309: 2039
40: 173	94: 491	148: 857	202: 1231	256: 1619	310: 2053
41: 179	95: 499	149: 859	203: 1237	257: 1621	311: 2063
42: 181	96: 503	150: 863	204: 1249	258: 1627	312: 2069
43: 191	97: 509	151: 877	205: 1259	259: 1637	313: 2081
44: 193	98: 521	152: 881	206: 1277	260: 1657	314: 2083
45: 197	99: 523	153: 883	207: 1279	261: 1663	315: 2087
46: 199	100: 541	154: 887	208: 1283	262: 1667	316: 2089
47: 211	101: 547	155: 907	209: 1289	263: 1669	317: 2099
48: 223	102: 557	156: 911	210: 1291	264: 1693	318: 2111
49: 227	103: 563	157: 919	211: 1297	265: 1697	319: 2113
50: 229	104: 569	158: 929	212: 1301	266: 1699	320: 2129
51: 233	105: 571	159: 937	213: 1303	267: 1709	321: 2131
52: 239	106: 577	160: 941	214: 1307	268: 1721	322: 2137
53: 241	107: 587	161: 947	215: 1319	269: 1723	
54: 251	108: 593	162: 953	216: 1321	270: 1733	

Correction exercice 1

Bonjour

```
#include <stdio.h>
#include <conio.h>
char nom[10],c;
main()
{
puts("Quel est votre nom ? ");
scanf("%s",nom);
printf("\nBonjour\t%s",nom);
c=getch();
return (0);
}
```

Correction exercice 2

Périmètre

```
#include <stdio.h>
float perim_rect (float lon, float larg)  /* ici pas de ; !! */
{
        float perimetre;              /* variable locale … la fonction */
        perimetre = 2*(lon + larg);
        return perimetre;             /* c'est ainsi que l'on renvoi le résultat de la fonction.*/
}

float alt_perim_rect (float lon, float lar)
{
        return 2*(lon+lar);
}
main()
{
        float L,l;                            /* déclaration de 2 variables globales dans main */
        printf ("\n Entrer la longueur ");
        scanf ("%f", &L);
        printf ("\n Entrer la largeur ");
        scanf ("%f", &l);
        printf ("\n\t Le p,rimetre est : %f ", perim_rect (L,l));
        /* C'est dans la fonction printf, comme paramètre que l'on appelle la
        perim_rect */
}
```

Correction exercice 5

RC

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float r,c,vf,tau,t;
main()
{
puts("\nCalcul de vs=vi*(1-e-(t/tau)) lors de la charge de R dans R  avec vs(t0)=0");_
puts("pour 20 valeurs de t comprises entre 0 et 5*tau");
puts("Entez R C et vf (tension finale) séparer les entrées par un espace: ");
scanf("%f %f %f",&r,&c,&vf);
tau=r*c;
for(t=0;t<=5*tau;t+=tau/4) printf("\nà t= %2f s -> vs= %f V",t,vf*(1-exp(-t/tau)));
puts("\ntapez une touche pour continuer");
c=getch();
return (0);
}
```

Correction exercice 6

Jeux

```
#include <stdio.h>
#include <stdlib.h> /* pour rand() */
```

```
#include <time.h>   /* pour trouver l'heure pour srand */
void main(void)
 {
  int solution,reponse,nb_essais=0;
  {time_t t;srand((unsigned) time(&t)); } /* initialiser le générateur à partir du compteur de temps, pour
                qu'il soit plus aléatoire */
  solution=rand()%11;    /* reste sera toujours entre 0 et 10 */
  do
   {
    nb_essais++;
    puts("prOpésez votre nombre entre 0 et 10");
   scanf("%d",&reponse);
   }
  while (reponse!=solution);
  printf("trouvé en %d essais\n",nb_essais);
 }
```

Correction exercice 7

Moyenne

```
#include <stdio.h>
void main(void)
 {
  int i,N;
  float note,somme=0,moyenne;
  puts("nombre de notes ? ");
  scanf("%d",&N);
  for(i=0;i<N;i++)
   {
    printf("entrez votre %dième note",i+1);
   scanf("%f",&note);
   somme+=note;
   }
  moyenne=somme/N;
  printf("moyenne calculée :%5.2f\n",moyenne);
 }
```

Correction exercice 8

Puissance

```
#include <stdio.h>
void main(void)
 {
  int puissance=1,max;
  puts("nombre maximal désiré (ne pas dépasser 16000) ?");
  scanf("%d",&max);
  while (puissance<max) printf("%d\n",puissance*=2);
 }
```

Correction exercice 9

Calculatrice

```
#include <stdio.h>
void main(void)
 {
  float val1,val2,res;
  char Opéval2
  int fin=0;
  do
   {
    puts("calcul à effectuer (par ex 5*2), ou 1=1 pour finir ? ");
   scanf("%f%c%f",&val1,&Opé&val2);
   switch (Opé
    {
```

```
      case '*':res=val1*val2;break;
      case '/':res=val1/val2;break;
      case '+':res=val1+val2;break;
      case '-':res=val1-val2;break;
      case '=':fin++; /* pas besoin de break, je suis déjà au } */
     }
    if (!fin) printf("%f%c%f=%f\n",val1,Opéval2,res);
   }
  while (!fin);
 }
```

Correction exercice 10

Nombres premiers (voir document annexe)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>   /* pour div*/

void touche(void)     /* attend une touche */
{
 char c;
 puts("\ntapez sur une touche");
 c=getch();
}


void main()
{
div_t x;
int i,j,max;
char nonprem;

puts("Nombre max du calcul : ");
scanf("%i",&max);
printf("\nCalcul des %i premiers nombres premiers\n 1",max);
for(i=3;i<max;i++)
{       nonprem=0;
        for(j=2;j<i;j++)
        {
                x=div(i,j);
                if (x.rem==0)   /* voir fonction div dans stdlib */
                {
                nonprem++;
                break;
                }
        }
        if (nonprem==0) printf("%7.i ",i);
}
touche();
return (0);
}
```

Correction exercice 11

Jeux des allumettes

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>   /* pour div*/

int nbal,max;
char couppc;

void touche(void)     /* attend une touche */
{
```

```
 char c;
 puts("\ntapez sur une touche");
 c=getch();
}

void affiche()  /* affiche en semi graphique le nombre d'allumettes restant*/
{
int i;
for(i=1;i<=nbal;i++)
                {
                putchar(178);
                putchar(' ');
                }
putchar('\n');
for(i=1;i<=nbal;i++)
                {
                putchar(177);
                putchar(' ');
                }
putchar('\n');
for(i=1;i<=nbal;i++)
                {
                putchar(177);
                putchar(' ');
                }
putchar('\n');
}

int pc()        /* c'est au PC de jouer*/
{
int ote,m;
affiche();
m=max+1;
ote=nbal-(((nbal-1)/m)*m+1);   /* tout se joue ici !!!!! */
if (ote<=0) ote=1;
printf("J ôte %i allumette",ote);
if (ote>1) puts("s");
nbal-=ote;
touche();
couppc=1;
return(nbal);
}

int joueur()  /* c'est au joueur de jouer*/
{
int ote;
affiche();
do
{
puts("combien ôtez vous d'allumettes ? ");
scanf("%i",&ote);
}
while (ote>max);
nbal-=ote;
couppc=0;
return(nbal);
}

void resultat() /* on affiche le vainqueur*/
{
affiche();
```

```
if (couppc==1) puts("j'ai gagné"); else puts("t'as gagné");
}

main()
{
clrscr();
puts("\nJeu des allumettes, il ne faut pas prendre la dernière mais au mois une\n) ;
puts("Combien d'allumettes au départ");
scanf("%i",&nbal);
puts("combien d'allumettes peut on prendre au maximum à chaque tour");
scanf("%i",&max);
while((joueur()>1)&&(pc()>1)) ;
resultat();
touche();
return(0);
}
```

Structure d'un programme en C

- Déclaration des bibliothèques de fonctions (*.h) à lier avec le programme, (les fichier *.h contiennent les prototypes des fonctions qui seront liées automatiquement au programme)
  Si la bibliothèque n'est pas dans le répertoire par défaut, son nom et chemin sera placé entre et non entre < >
- Déclaration des constantes (const) en ROM
- Déclaration des variables globales en RAM, accessibles depuis n'importe quel point du programme
- Fonctions : une fonction possède des paramètres d'entrée et des variables de sortie, elles effectue un certain nombre de tache. Elle peut disposer de variables locales (stockées temporairement dans une pile) qui lui sont propres
- Programme principal : fonction main, la fonction main est obligatoire, elle représente les points d'entrée et de sortie d'un programme en C

Exemple : TVADEMO.C calcul du prix TTC à partir du prix HT et de la TVA

```
                                        /* indique un commentaire
/* CD 06/1999 */
/* TVADEMO.C calcul de la valeur TTC d'un prix HT, version longue */
/* bibliothèques à lier avec le programme */

#include <stdio.h>    /* gets, puts, printf */
#include <stdlib.h>   /* atof */

const float tva=20.6
float prixht;

/*****************************************************************/

float calcul(float p)  /*une variable locale p est créee*/
{
        return(p*(1+tva/100));
}

/*****************************************************************/

void demande_prix_ht(void)
{
char sprix[5];

        puts("\ndonnez le prix Hors Taxes (0 pour finir):");
        gets(sprix);
        prixht=atof(sprix);
}

/*****************************************************************/

void affiche_prix_ttc(void)
{
float r;    /* variable locale, créée dans la pile et détruite à la sortie de la fonction */
        r=calcul(prixht);
        printf("le prix TTC est %f: ",r);
}

/*****************************************************************/

/* ici commence le programme principal*/
void main(void)  /* pas de paramètre en entrée, rien en sortie */
{
        printf("\nCalcul du prix TTC, la TVA étant de %f\n",tva);
        do
        {
                demande_prix_ht();
                affiche_prix_ttc();
        }
        while(prixht!=0);
}
```

Indique une bibliothèque à inclure, <> pour une bibliothèque dans le répertoire par défaut

Déclaration d'un constante qui sera créée en ROM, ici une constante réelle

Déclaration d'une variable globale qui sera rangée en RAM, ici une variable réelle

Fonction avec un paramètre réel p en entrée (placé dans la pile et une variable réelle en sortie, le corps de la fonction est compris entre deux accolades.

Void indique une absence de paramètre (vide)

Déclaration d'une variable locale à la fonction demande_prix_ht, ici une chaine de 5 octets

Puts, gets, atof et printf sont des fonctions déclarées dans stdio et stdlib, dans le cas contraire le compilateur génère un message d'erreur

%f indique d'afficher la variable r sous la forme d'un réel

\n représente un passage à la ligne

La boucle do while sera exécutée (instructions entre accolades) tant que prixht sera non nul

➲ Le C permet d'écrire des programmes très condensés et évite de déclarer des variables temporaires consommatrice de RAM et de temps

Le programme TVA.C une fois condensé

```
/* DEMO.C calcul de la valeur TTC d'un prix HT, version condensée */
#include <stdio.h>    /* gets, puts, printf */
#include <stdlib.h>   /* atof */
const float tva=20.6; /* tva est rangé en ROM sous le format float */
void main(void)       /* programme principal */
{char sprix[5];       /* variables locales placées dans une pile en RAM*/
float r;
  printf("\nCalcul du prix TTC, la TVA ,tant de %f\n",tva); /* \n pour passer à la ligne */
  do
  {         puts("\ndonnez le prix Hors Taxes (0 pour finir):");
            printf("le prix TTC est %f: ",(r=atof(gets(sprix)))*(1+tva/100)); /* c'est peu clair !!!*/
  }
  while(r!=0);
}
```

Remarque : la constante tva aurait pu être ici remplacé par une **directive #define** , dans ce cas la valeur de tva ne se situe pas dans la mémoire ROM mais est remplacée par 20.6 chaque fois qu'elle apparaît (comme la directive equ en assembleur)
#define nom valeur : remplace chaque occurrence du nom par la valeur. **exemple** : #define PI 3.1415926

Les variables

Types et tailles (sur CC11LITE)

unsigned char 8 bits
'a' correspond à un octet, "a" à deux ('a' et \0).:
\ n nouvelle ligne
\ t tabulation
\ b backspace
\ r retour chariot (même ligne)
\ f form feed (nouvelle page)
\ ' apostrophe
\ \ antislash
\ " double quote
\ 0 nul
\ nombre en octal sur 3 chiffres
\ xnombre : en hexa
signed char 8 bits (ou char)
unsigned short 16 bits
signed short 16 bits
unsigned int 16 bits
signed int 16 bits
unsigned long 16 bits
signed long 16 bits
float 32 bit, exposant 8 bits, mantisse 24 bits
double 32 bit, exposant 8 bits, mantisse 24 bits
pointeur 16 bits

➲ Déclarées en début de programme elles sont **globales**, accessibles depuis n'importe qu'elle fonctions, rangées en RAM
➲ Déclarées dans une fonction, elles sont **locales**, créées en RAM mais dans une pile, la mémoire utilisées est libérée lors de la sortie de la fonction.
➲ Déclarées dans une fonction avec le type **static**, elles sont **locales** mais rangées de manière statique en RAM (elles ne sont " visibles " que dans la fonction ou elles ont été déclarées)

Une variable peut être initialisée lors de sa déclaration, int i=1234 ;
Un tableau est un ensemble de variables indicées de même type, ex :

```
int chiffres[]={0,1,2,3,4,5,6,7,8,9};
int tableau[20]={1,2,3}; /* les 17 autres à 0 */
char mess[]="bonjour"; /* évite de mettre ={'b','o',.,'r',\0} */ le dernier indice d'une chaîne est toujours 0,
```

Pointeurs : un pointeur représente l'adresse d'une variable ex : int *p ; indique que p est un pointeur sur un entier.

```
char *porta;        /* pointeur 16 bits sur un octet */
porta  = 0x1000;    /* initialisation du pointeur    */
*porta = 0x80;      /* placer $80 … l'adresse $1000  */
```

Formats des types sur printf

- ❑ c (char)
- ❑ s (chaîne de caractères, jusqu'au \0)
- ❑ d (int)
- ❑ u (entier non signé)
- ❑ x ou X (entier affiché en hexadécimal)
- ❑ f (réel en virgule fixe)
- ❑ p (pointeur)
- ❑ % (pour afficher le signe %).
- ❑ \t (tabulation), \n (retour à la ligne), \\ (signe \), \nb tout code ASCII, en décimal, hexa ou octal (\32=\040=\0x20=' ').

Changement de type : Cast

Int i ;Float f
I=(int)(f) i est égal à la partie entière de f
F=(float)(i) f est égale à i, la partie décimal étant égale à 0

Les constantes

Déclarées comme les variables, mais avec le type const, elles sont rangées dans la ROM. Ex : const char a,b,c ;

Les tests

La condition if : *si alors*

```
if (a==2)
        {instructions….     }
esle
        {instructions…      }
```

Le test switch : Switch – case - break
Test de c

```
switch(c)
 {   case 'a': { voyelle=1 ; break ; }
  case 'e': { voyelle=1 ; break ; }
  case 'i': { voyelle=1 ; break ; }
  case 'o': { voyelle=1 ; break ; }
  case 'u': { voyelle=1 ; break ; }
  case 'y': { voyelle=1 ; break ; }
default :voyelle=0 ;  }
```

Boucles

Boucles for : *Pour i=0 et tant que i <10 incrémenter i à chaque boucle :*

```
For(i=0 ;i<10 ;i++)
        {instructions…      }
```

effectuera 10 fois la boucle d'instructions

Boucle while : *Tant que i<10 effectuer la bouche (la boucle n'est pas effectuée si i est supérieur ou égale à 10)*

```
While (i<10)
        {instructions…      }
```

Boucle do while : *Tant que i<10 effectuer la bouche (la boucle est pas effectuée au moins une fois)*

```
Do
        {instructions…      } while(i<10) ;
```

Expressions / opérateurs

Leur hiérarchie (priorité) est celle définie dans ANSI. La liste ci-dessous est classée par ordre de priorité décroissante.

Fonctions	Opérateur	Description	Exemples
Identificateurs	**()**	Appel de fonction	
	[]	Indice de tableau	tableau[3]=5;
opérateurs unaires	**!**	Négation logique (NOT)	b=!a;
	~	Complément binaire (1)	b=~a
	-	Moins unaire	b=-a;
	+	Plus unaire	b=+a;
	++	Préincrément ou postincrément	b=a++; (b=a puis a=a+1)
	--	Prédécrément or postdécrément	b=a--; (b=a puis a=a-1)
	&	Adresse de	b=&a; (b égale l'adresse de a)
	*****	Indirection (adressage indexé)	b=*a; (b=contenu de l'adresse de a)
opérateurs binaires	*****	Multiplication	c=a*b;
	/	Division	c=a/b;
	+	Plus binaire	c=a+b;
	-	Moins binaire	c=a-b;
	<<	Décalége à gauche	c=a<<b; (a est décalé b fois à gauche)
	>>	Décalége à droite	c=a>>b; (a est décalé b fois à droite)
	&	ET entre bits	c= a & b; (ET logique bit à bit)
	^	OU exclusif entre bits	c= a ^b;
	|	OU entre bits	c= a|b;
Tests	**<**	Strictement inférieur	if a < b
	<=	Inférieur ou égal	if a >= b
	>	Strictement supérieur	if a > b
	>=	Supérieur ou égal	if a >= b
	==	Egal	if a ==b (si a est égale à b)
	!=	Différent	if a != b
	&&	ET logique	if ((a=5) && (b=2))
	||	OU logique	if ((a=5) ||(b=2))
	?:	condition	z=(a>b)?a:b (Si a>b a z=a sinon z=b)
Affectation	**=**	Affectation simple	a=b; (a prend la valeur de b)
Auto-affectations	***=**	Affectation produit	a*=2 (a=a*2)
	/=	Affectation quotient	a/=2 (a= a/2)
	%=	Affectation reste	a%=2 (a= le reste de a/2)
	+=	Affectation somme	a+=2 (a=a+2)
	-=	Affectation différence	a-=2 (a=a-2)
	&=	Affectation ET entre bits	a&=5 (a=a&5)
	^=	Affectation OU EX entre bits	a^=5 (a=a^5)
	|=	Affectation OU entre bits	a|==5 (a=a|=5)
	<<=	Affectation décalége gauche	a<<=5 (a=a<<5)
	>>=	Affectation décalége droite	a>>=5 (a=a>>5)

conditionnel ? :
C'est un (le seul) opérateur ternaire. L'expression a?b:c vaut la valeur de b si a est vrai (entier, différent de 0), et c si a est faux.
Exemple : max=a>b?a:b

Bibliothèques CANSI (réduites)

Stdio.h
getc getchar gets getw printf putc putchar puts scanf printf
conio.h
cgets clreol clrscr cprintf cputs cscanf delline getch getche gotoxy insline inp inport inportb inpw kbhit outp outport outportb outpw putch puttext _setcursortype textattr textbackground textcolor textmode wherex wherey window
ctype.h
isalnum isalpha isascii iscntrl isdigit isgraph islower isprint ispunct isspace isupper isxdigit toascii tolower toupper
math.h
abs acos, acosl asin, asinl atan, atanl atan2, atan2l atof, _atold cabs, cabsl ceil, ceill cos, cosl cosh, coshl exp, expl fabs, fabsl floor, floorl fmod, fmodl frexp, frexpl hypot, hypotl labs ldexp, ldexpl log, logl log10, log101 matherr, _matherrl modf, modfl poly, polyl pow, powl pow10, pow10l sin, sinl sinh, sinhl sqrt, sqrtl tan, tanl tanh, tanhl
string.h
memccpy memchr memcmp memcpy memicmp memmove memset movedata movmem setmem stpcpy strcat strchr strcmp strcmpi strcpy strcspn strdup _strerror strerror stricmp strlen strlwr strncat strncmp strncmpi strncpy strnicmp strnset strpbrk strrchr strrev strset strspn strstr strtok strxfrm strupr

Bibliothèques CC11

Hc11.h : Ce fichier contient des déclarations pour les registres du 68HC11 standard. Le fichier contient aussi des déclarations pour le X68C75 qui se trouve sur le Controlboy 2 et 3. Pour utiliser ces déclarations, écrire :
#define Controlboy3
#include <hc11.h>

- **abort.c** : void abort() : Sortie inconditionnelle d'une fonction
- **abs.c** : int abs(int i) : retourne la valeur absolue d'un entier
- **aloc.c**
  void *calloc(size_t nelem, size_t size) : retourne un pointeur sur un espace de mémoire réservé à un tableau de nelem, tous de taille size. La mémoire allouée est initialisée avec des zéros.
  void *malloc(size_t size) : : retourne un pointeur sur un espace de mémoire réservé à un objet de taille size. La mémoire allouée n'est pas initialisés
  void *realloc(void *p, size_t size) : Change en size la taille de l'objet pointé par p. Si la nouvelle taille est plus petite que l'ancienne, seul le début du contenu de l'objet est conservé. Realloc retourne un pointeur sur le nouvel espace de mémoire
  void free (void *p) libère l'espace mémoire pointé par p
  Pour que ces fonctions marche correctement, il faut initialiser un tas avant.
- **int atoi(const char *s)** : long atol(const char *s) convertit une chaine ASCII en integer ou float
- **Ctype.C** static const char _ctype[1+256] : définition des 256 caractères ASCII
  int tolower(int c) : convertit majuscules en minuscules
  int toupper(int c) : convertit minuscules en majuscules
- **Int32.c** librairie pour les opérations sur entiers de 32bits
  void uint32_inc(INT32 *psrc) : incrémentation

void uint32_dec(INT32 *psrc) décrémentation
INT32 int32_add(INT32 left, INT32 right) addition
INT32 int32_sub(INT32 left, INT32 right) soustraction
INT32 int_to_int32(int i)
INT32 uint_to_int32(unsigned int i) non signé vers signé
INT32 int32_add_int(INT32 left, int right)
INT32 int32_sub_int(INT32 left, int right)
INT32 int32_add_uint(INT32 left, unsigned int right)
INT32 int32_sub_uint(INT32 left, unsigned int right)

- **Iosci.c**
  Int Getsci() : attend et retourne le caractère sur SCI
  int putsci(char) : Emet le caractère sur SCI
  int getstsci(char *p) : lit une chaîne (finissant par 0) et la place dans le pointeur p
  int putstsci(char *p) : émet la chaîne pointé par p
- **Iosciit.c** gestion du SCI en interruption
- **lib.c** void _assert(char *s, char *f, int l)
- **memchr.c** : void *memchr(void *p, int c, size_t n) : recherche c entre p et p+n
- **memcmp.c :** int memcmp(void *s1, void *s2, size_t n) : recherche si les bloc (s1 à s1+n) et (s2 à s2+n) sont idendentiques.
- **memcpy.c :** void *memcpy(void *s1, void *s2, size_t n) : s1 devient égale à s2
- **memmove.c** : void *memmove(void *s1, void *s2, size_t n) : deplace s2 vers s1
- **memset.c** : void *memset(void *p, int c, size_t n) : remplit p à p+n avec c
- **modf.c** : double modf(double v, double *ip) : voir math.h
- **newheap.c** :void _NewHeap(void *start, void *end) : réserve mémoire
- **print.c**
  Printf() reconnaît les directives suivantes

%d	nombre entier décimal signé
%o	nombre entier octal non signé
%x	nombre entier hexadécimal non signé
%u	nombre entier décimal non signé
%s	châine de caractères
%c	caractère ASCII
%f	nombre virgule flottante

Pour imprimer un nombre en virgule flottante, il faut compiler le fichier print.c avec l'option -DFLOAT_PRINTF et il faut ajouter le fichier float.a11.

char *gets(char *s) : lit un caractère sur (le clavier sur CBOY3)
void _itoa(char *buf, unsigned int i, int base) : convertit entier en ascii
int _print(char **ps, void (*_put)(char **, int), char *fmt, va_list va)
int printf(char *fmt, ...) : voir ci dessus
int fprintf(FILE *fp, char *fmt, ...)
int puts(const char *s) envoie une chaine vers la sortie
int sprintf(char *s, char *fmt, ...) : ecrit dans la chaine s
size_t strlen(const char *s) : retourne la longueur de la chaine s

- **puts.c :** int puts(const char *s) : envoie une chaine vers la sortie standard (putc.c)
- **rand.c** : void srand(unsigned int seed) int rand(void) : retourne un entier aléatoire
- **rdsci.c** : unsigned int read_sci() : lit un caractère sur SCI (sans IT)
- **rdspi.c** : unsigned char read_spi() : lit un caractère sur SCI (sans IT)
- **readatod.c** : unsigned char read_atod(int index) : lit un octet sur le CAN index
- **serial.c** : void setbaud(BaudRate baud) : change la vitesse du SCI !
- **sprintf.c** : int sprintf(char *s, char *fmt, ...) : printf mais le résultat est dans la chaine s
- **stdarg.c** : char *_va_start(char *p_rts, void *p_arg, int size)
- **stdio.c** : (idem à print.c moins quelques fonctions)
  char *gets(char *s)
  void _itoa(char *buf, unsigned int i, int base)
  int _print(char **ps, void (*_put)(char **, int), char *fmt, va_list va)

Opérations sur les chaines de caractères (fonctions commencant par str:

- **strcat.c** : char *strcat(char *s1, const char *s2)
- **strchr.c** : char *strchr(const char *s, int c)
- **strcmp.c** : int strcmp(const char *s1, const char *s2)
- **strcspn.c** : size_t strcspn(const char *s1, const char *s2)
- **strlen.c** : size_t strlen(const char *s)
- **strncat.c** : char *strncat(char *s1, const char *s2, size_t n)
- **strncmp.c** : int strncmp(const char *s1, const char *s2, size_t n)
- **strncpy.c** : char *strncpy(char *s1, const char *s2, size_t n)
- **strpbrk.c** : char *strpbrk(const char *s1, const char *s2)
- **strrchr.c** : char *strrchr(const char *s, int c)
- **strspn.c** : size_t strspn(const char *s1, const char *s2)
- **strstr.c** : char *strstr(const char *s1, const char *s 2)
- **strtol.c** : long strtol(const char *s, char **end, int base)
- **strtool.c** : unsigned long strtoul(const char *s, char **end, int base)

Divers

- **tozerpag.c** : void to_zero_page(PFN start, PFN end)
- **wreeprom.c** : void write_eeprom(unsigned char *addr, unsigned char c) : écrit c à l'adresse EEPROM addr
- **wrsci.c** : void write_sci(unsigned int i) : écrit i sur SCI
- **wrspi.c** : void write_spi(unsigned char c) : écrit c sur SPI

Librairies en assembleur 68HC11

- **Diverses librairies A11**

INT32.A11 Addition et soustraction d'integer 32bits
CRTA11 initialisation , pour les integers : addition, soustraction, multiplication, division , décalages et strcpy
ENDA.11 clôture

- **Pour les entrées/sorties**

CLAVIER.A11 int keyget() , remplace getchar, retourne 0 ou le code ASCII de la touche enfoncée
LCD.A11 lcdinit() et **putchar()**
LCDCLAV.C : void lcdint(), void gotox(char), char keyget() ; void putchar(char)

- **Pour les nombres réels (float) :**

FLOAT.A11 addition, soustraction, multiplication, division, ftoa, ftoi,itof
FP. A11 librairie mathématique
FPSTRFLT.A11 ascii vers float
FP2INT.A11 float vers integer
FPFLTSTR.A11 float vers ascii
FPINTFLT.A11 uinteger vers float et integer vers float
FPCMP.A11 comparaison de deux float
FPADD.A11 addition de deux float
FPSUB.A11 soustraction de deux float
FPMODMUL.A11 multiplication de float (240uS à Q=8MHz)
FPDIV.A11 division de deux float
FPABS.A11 retourne la valeur absolue d'un float
FPEXP.A11 exp10, exp, puissance n
FPLOG.A11 log10, ln
FPSQRT.A11 racine carrée d'un float (8177uS à Q=8MHz)
FPTRIG.A11 atan, sin, cos, tan, degrés/radians, radians/degrés

- **Gestion des interruptions sur CC11 controlboy:**

```
/*Générer un signal carré de 2.15 KHz sur PA3  */
#define controlboy3
#include <hc11.h>
#define t232us 464 /* 464 cycles pour 232uS avec Q=8Mhz*/

#pragma interrupt_handler oc5Int   /*oc5Int se terminera par RTI*/
void oc5Int()                      /*routine d'interruption*/
{
        TOC5 +=t232us;       /*durée de l'interruption = 232us*/
        TFLG1 |= 0x8;    /*on met oc5 au niveau haut (drapeau)*/
}

#pragma abs_address:0xFFE0      /*vecteur routine d'IT */
void (*oc5int_vector)() = oc5Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address

void main(void)
{       PACTL |=0x08;      /*Timer oc5 activé et PA3 en sortie*/
        TMSK1 |=0x08; /*IT de oc5 activée et PA3 bascule à "0"*/
        TCTL1 |=1;   /*action périodique définie sur le port A*/
        TCTL1 &=0xfd;                /*OMx="0" et OLx="1"*/
        asm(" cli ");          /*autoriser les interruptions/
        for(;;);
```

Notes :