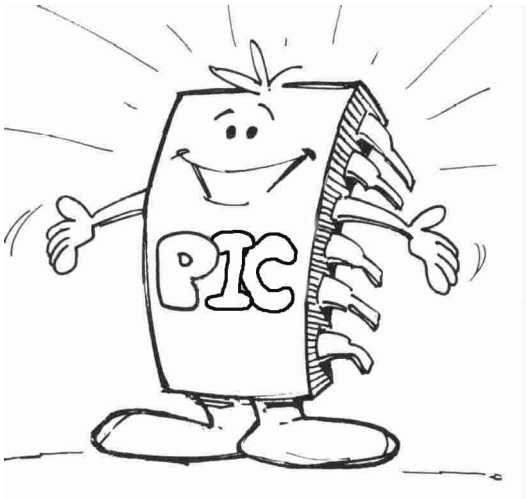
C

MINIMUM



connaissances nécessaires à la programmation des microcontrôleurs PIC18 en langage C

(une introduction au langage c A.N.S.I maj n°4)

Equipe de formation sur les microcontrôleurs PIC

Robert Toquebeuf
Lycée Adam de Craponne
13700 Salon de Provence
Académie d'Aix-Marseille
robert.toquebeuf@laposte.net

Christian Dupaty
Lycée Fourcade
13120 Gardanne
Académie d'Aix-Marseille
c.dupaty@aix-mrs.iufm.fr

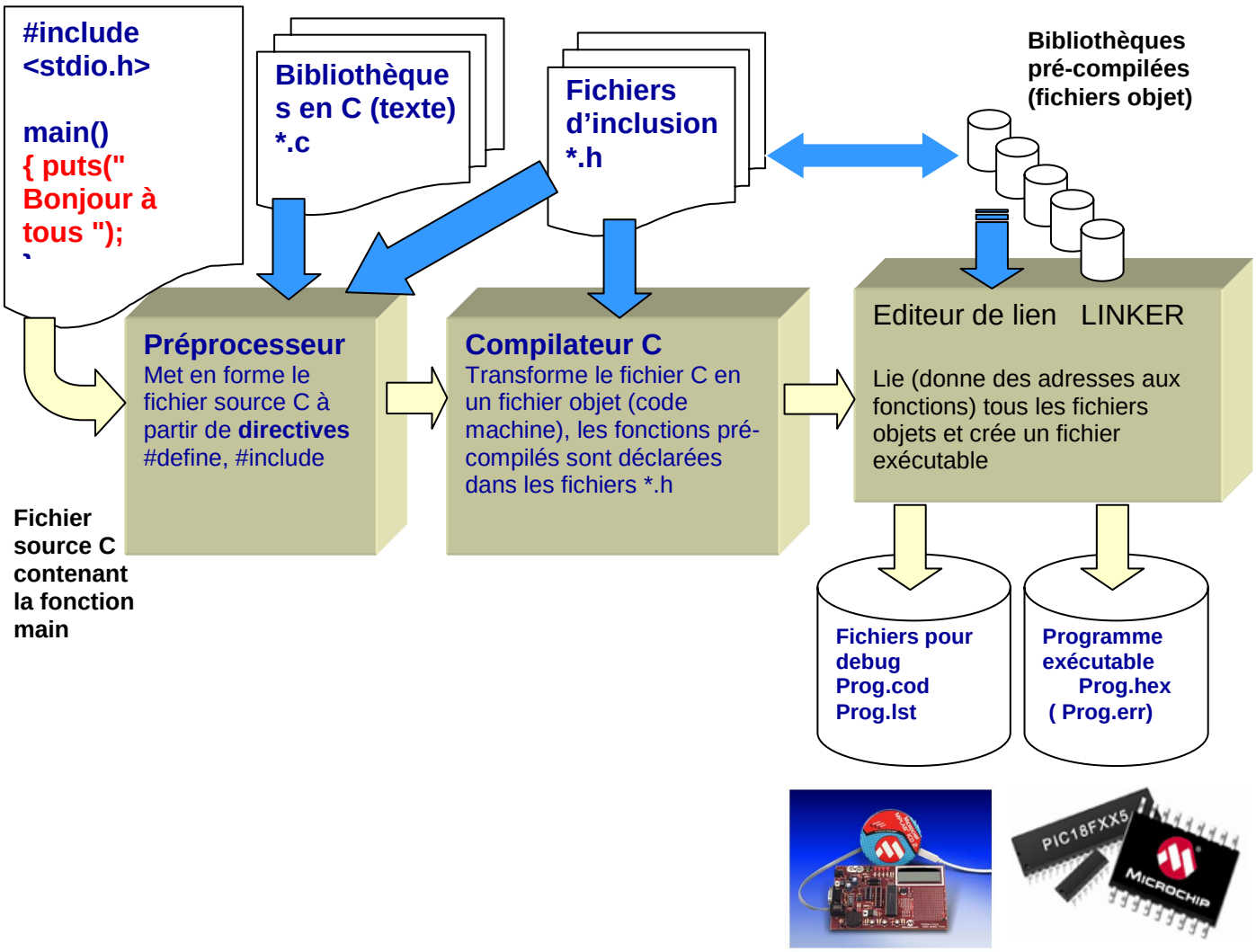
SOMMAIRE

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



Le langage C est un langage de programmation évolué, typé, modulaire et structuré :

- Evolué : Le code est indépendant du processeur utilisé
- Typé : Un type est l'ensemble des valeurs que peut prendre une variable
- Entiers, réels, caractères etc … ou à définir par le programmeur
- Modulaire et structuré :Tout programme est décomposable en tâches simples (3 structures algorithmiques de base) qui seront regroupées sous forme de modules (fonctions) qui eux même regroupés de façon cohérente en tâches plus complexes (structurés) formeront le programme.

2. LA SYNTAXE DU C : le premier programme

```
#include <p18f452.h>

#define duree 10000

char c;
float pht;

void tempo(unsigned int count);

void main(void)
{
     PORTB = 0x00;
     TRISB = 0x00;
     while(1) {
          PORTB++;
          tempo(duree);
          }
}

void tempo(unsigned int compte)
{
     while(compte--);
}
```

Bibliothèque

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

Variables globales (inutiles ici):
char : octet
float réel

Prototype de la fonction tempo, indispensable car le corps de celle est à la fin du programme

Représentation des nombres :
12 codé en décimal représente 12
0xC codé en hexadécimal représente 12
0b00001100 codé en binaire représente 12

Boucle infinie incrémentant PRB

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

main : fonction principale et point d'entrée du programme.
void indique qu'il n'y pas de paramètre d'entrée.

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 constante (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. 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 gauch
	>>	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=l
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)

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

5. Boucles

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

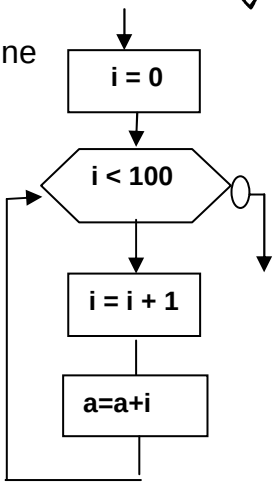
Ex **:** **char i,j=0;**

```
for (i=0 ;i<100 ;i++)
        {
                j=j+i;
        }
```

Condition de départ

Condition de fin

Evolution de la variable



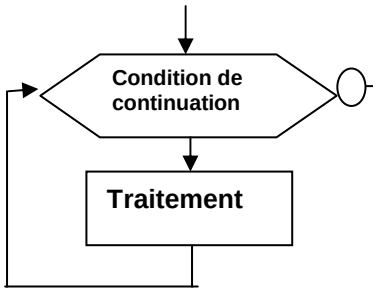
(dans cet exemple les accolades sont superflues, il n'y a qu'une instruction dans la boucle)

```
char j=0,i=20 ;
for (;i<100 ;i++) j=j+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;
j=0;
while (i<100)
        {
                j=j+i;
                i++ ;
        }
```

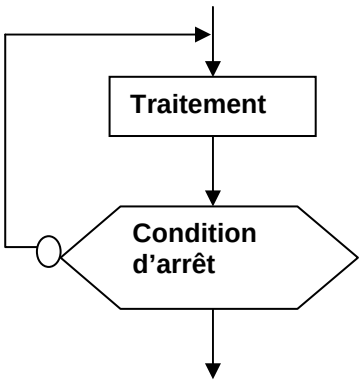
Tant que **« condition de continuation »** faire **« traitement »**



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

```
do
        {
                j=j+i;
                i++ ;
        }
while (i<100)
```

Répéter « traitement » **jusqu'à** « condition d'arrêt ».



6. Branchements conditionnels :

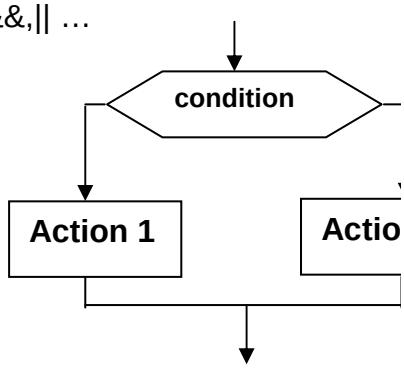
if else

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 : ==, <,>,<=,>=, !=,&&,|| …

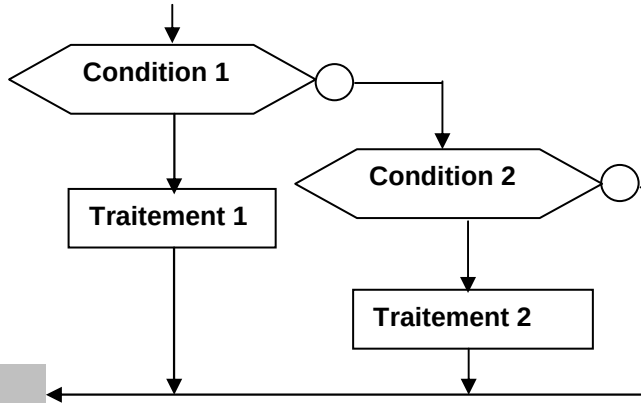
Si « condition »**alors** « actio n 1 » **sinon** « action 2 »



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);
   }
}
```

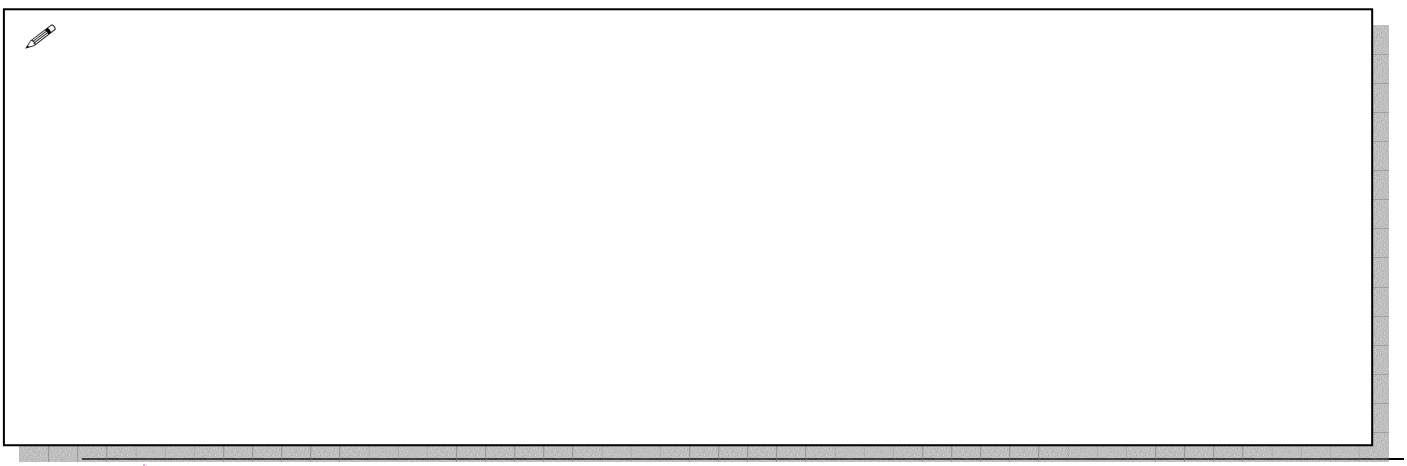


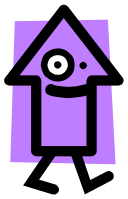
Selon cas faire :
Cas1 : « traitement 1 »
Cas2 : « traitement 2 »
Cas3 : « traitement 3 »
etc…

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)





7. Les pointeurs

- Ce sont des **variables** particulières **qui contiennent l'adresse d'une variable**, elles ont toujours le même format, sur PIC18 un pointeur est une valeur sur 16bits ou 20 bits.
- 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.

8. Tableaux

Un tableau est un regroupement dans une même variable de variables de même type
int chiffres[ ]={10,11,12,13,14,15,16,17,18,19}
/* un tableau de 10 entiers*/
chiffre[0]=10, et chiffre[3]=13

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

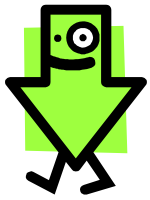
Les chaînes de caractères sont des tableaux de caractères finissant par 0, *une chaîne est entourée de " et est automatiquement terminée par \0*
char message[]= "bonjour"; est la même chose que
char message[]={'b','o','n','j','o','u','r',\0} ;

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 *)(0xF80) ex: var=PORTA
la valeur 0xF80 est transformée en un pointeur sur un char. PORTA est équivalent au contenu de ce pointeur, donc au contenu de l'adresse 0xF80

9. 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++)  ;
}
```

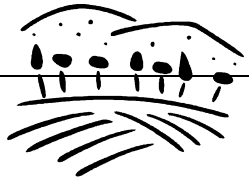
10. Structures

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,classe[i].a
ge,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	RBO

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 ;
```

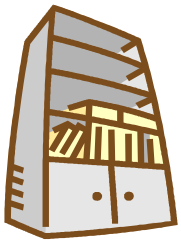
```
PORTEbits.RE0
PORTEbits.ALE
PORTEbits.AN5
Partagent le même bit physique
```

Attention, cette bibliothèque standard n'est pas complète pas sur MCC18

13. Bibliothèque standard stdio.h

- 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 …)*

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

14. 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;
}
```

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

16. Exercices sur PC:

Ces exercices sont à réaliser avec un compilateur C sur PC, de type TURBO C

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(void)        /* 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(void)  /* 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);
}
```

Notes :