LE LANGAGE C

1- Historique

Le langage C a vu le jour au début des années 1970 aux Bell Laboratories (AT&T) en parallèle avec le système d'exploitation UNIX. Le langage C a été défini dans l'ouvrage THE C PROGRAMMING LANGUAGE de Brian W. Kernighan et Dennis M. Ritchie.
Il est aujourd'hui normalisé (1989) ce qui assure une grande portabilité aux logiciels écrits dans ce langage. Le langage C normalisé est qualifié du sigle de C-ANSI (American National Standards Institute). Le langage C est un langage de programmation évolué, a priori indépendant de la structure des ordinateurs, mais permettant d'accéder à toutes les ressources d'une machine.
Il existe une évolution très importante du langage C, c'est le langage C++ qui emprunte presque tout au langage C, en particulier sa syntaxe, mais en lui rajoutant de nombreuses possibilités en termes de programmation orientée objet.

2- Du fichier source au fichier exécutable

Un programme écrit en langage C se présente généralement sous forme d'une ou plusieurs fonctions juxtaposées dans un fichier. C'est le fichier source saisi à l'aide d'un éditeur de texte, auquel on donne l'extension .c, exemple : essai.c. Ce dernier est traité par le compilateur qui génère un fichier objet, reconnaissable à son extension .obj.
Pour obtenir un fichier exécutable, il faut ensuite relier les différents fichiers objets nécessaires à l'aide d'un éditeur de liens (linker). Certaines fonctions appelées peuvent être regroupées dans des librairies ou fichiers objets.

Désignation des fichiers dans environnement AVR

*.c fichier source
*.obj fichier objet
*.h fichier entête
*.lib fichier librairie
*.hex fichier exécutable (Intel)

Fichiers nécessaires à la programmation ISP : *.rom, *.eep
Fichiers nécessaires à la fonction debugger : *.cof, *.rom, *.eep

Environnement DOS

*.c	fichier source
*.obj	fichier objet
*.h	fichier entête
*.lib	fichier librairie
*.exe	fichier exécutable

Environnement UNIX

*.c	fichier source
*.o	fichier objet
*.h	fichier entête
*.a	fichier librairie
*	fichier exécutable

3- Exemple de programme : allumer les leds du port B

```
#include <90s8535.h>                    Directive de pré-compilation
unsigned int i;                         Déclaration de la variable

void tempo(void)                        Définition d'une fonction ne renvoyant pas de
{                                       données (mot clé void)
        for (i=0;i<=60000;i++)          Boucle for (POUR)
        {
        ;
        }
}

void main(void)                         Programme principal (mot clé main)
{
        DDRB=0xff;                      Déclaration du port B en sortie
        PORTB=0xff;                     Ecrire des « 1 » sur le port B (allumer les leds)
        while(1)                        Boucle sans fin
        {
                PORTB=0x00;             Ecrire des « 0 » sur le port B (éteindre les leds)
                tempo();                Appel à la fonction « tempo »
                PORTB=0xff;             Ecrire des « 1 » sur le port B
                tempo();                Appel à la fonction « tempo »
        }
}
```

4- Descriptions des éléments d'un programme

Les directives de pré-compilation
Le rôle du pré-processeur ou pré-compilateur, est de réaliser des mise en forme et des aménagement de texte d'un fichier source, juste avant qu'il ne soit traité par le compilateur. Les instructions spécifiques pour indiquer quelles opérations effectuer durant cette étape de pré-compilation sont appelées directives. Elles commencent toutes par le caractère # et ne se terminent pas par un point-virgule.

La déclaration des variables
Tous les éléments utilisés doivent faire l'objet d'une définition préalable.

Les fonctions
En langage C, le seul type de sous-programme disponible est le type fonction. Toutes les fonctions sont externes les unes par rapport aux autres. L'ensemble des instructions composant une fonction sont groupées entre accolades ***{*** et ***}***. On dit que ces deux accolades ouvrante et fermante délimitent un bloc.
A priori, une fonction retourne obligatoirement une valeur et une seule. C'est l'instruction ***return*** qui permet de préciser quelle valeur doit être retournée. Il est possible de préciser explicitement qu'une fonction ne retourne aucune valeur à l'aide du mot clé ***void***.
Parmi toutes les fonctions, une et une seule, doit obligatoirement porter le nom ***main***; il s'agit de la fonction principale (en général à la fin du programme) par laquelle débute l'exécution.

Les commentaires
Toute séquence de caractères délimitée à l'aide de **/*** et ***/** est considérée comme un commentaire (également //).

5- Les données dans un programme

Les types de données

Type	*Taille (Bits)*	*Plage de valeur*	
bit	1	0 , 1	*(entier)*
char	8	-128 to 127	*(entier)*
unsigned char	8	0 to 255	*(entier)*
signed char	8	-128 to 127	*(entier)*
int	16	-32768 to 32767	*(entier)*
short int	16	-32768 to 32767	*(entier)*
unsigned int	16	0 to 65535	*(entier)*
signed int	16	-32768 to 32767	*(entier)*
long int	32	-2147483648 to 2147483647	*(entier)*
unsigned long int	32	0 to 4294967295	*(entier)*
signed long int	32	-2147483648 to 2147483647	*(entier)*
float	32	$\pm 1.175^{e}$-38 to ±3.402e38	*(réel)*
double	32	$\pm 1.175^{e}$-38 to ±3.402e38	*(réel)*

Les bases de représentation

Base	*Préfixe*	*Exemple*
Décimale	Aucun	29
Hexadécimale	0x	0xff
Octale	0	075
Binaire	0b	0b01010

Les constantes

Les constantes sont stockées en mémoire flash. On utilise l'attribut ***const*** pour en définir une.

Exemples :

```
const int integer_constant=1234+5;
const char char_constant='a';
const int integer_array2[10]={1,2};
```

Il est possible, à l'aide d'un suffixe, d'affecter à une valeur son type de donnée.

Type de constante	*Suffixe*	*Exemple*
Entier non signé (unsigned integer)	U	10000U
Entier long (long integer)	L	99L
Entier long non signé (unsigned long integer)	UL	99UL
Virgule flottante (floating point)	F	1.234F
Caractère (char)	' et '	'a'
Chaîne (string)	" et "	" Hello world "

Les variables

Syntaxe : <type> <identifiant>;

Exemples :

```
char a;
int b;
```

Les tableaux

Exemple 1 : tableau de valeurs

unsigned char Table[5]={1,2,3,4,5};

permet de créer un tableau (Table) contenant 5 données. Pour lire ce tableau il suffira de faire varier l'indice repérant la donnée à exploiter (ex : a = Table[i]).

Exemple 2 tableau de caractère ou chaîne de caractères

char Phrase[] = "hello world !";

Le type de tableau crée ici est en fait une chaîne de caractère. Le compilateur réserve ici un tableau de 14 octets, 13 pour hello world ! et 1 pour le caractère de fin de chaîne (caractère nul de code ASCII zéro).

Les pointeurs

Au lieu de travailler avec les noms des variables, il est également possible de travailler avec leurs adresses en mémoire. Tel est le but des pointeurs. Un pointeur est une variable qui contient l'adresse d'une autre variable.

Expression	***Description***
char *pt_a	Déclaration du pointeur pt_a destiné à désigner l'adresse d'une variable de type char.
*pt_a	Correspond à la valeur de la variable dont l'adresse est conservée par le pointeur pt_a.
pt_a	La variable pt_a étant un pointeur, elle conserve l'adresse d'une variable.
&var_a	Désigne l'adresse de la variable var_a.
Pt_a = &var_a	Affectation de l'adresse de la variable var_a au pointeur pt_a.
var_b = *pt_b	Affectation à la variable var_b de la valeur dont l'adresse est désignée par le pointeur pt_b.
*pt_b = var_b	Affectation de la variable var_b à la variable désignée par le pointeur pt_b.
pt_a = pt_b	Affectation au pointeur pt_a de l'adresse placée dans le pointeur pt_b.

6- Les opérateurs

Les opérateurs arithmétiques

Symbole	Opération
+	addition
-	soustraction
*	multiplication
/	division entière
%	reste de la division entière (cette opération n'est pas valable sur des réels)
a=b	affectation (a reçoit la valeur de b)

Les opérateurs arithmétiques raccourcis

Symbole	Opération
-=	notation contractée : a -= b donne a = a - b
+=	notation contractée : a += b donne a = a + b
*=	notation contractée : a *= b donne a = a * b
/=	notation contractée : a /= b donne a = a / b
--	post décrémentation : a=i-- . "a" reçoit la valeur de "i", puis "i" est décrémenté de 1. pré décrémentation : a=--i . "i" est décrémenté de 1 puis sa valeur est attribuée à "a"
++	post incrémentation : a=i++ . "a" reçoit la valeur de "i", puis "i" est incrémenté de 1. pré incrémentation : a=++i . "i" est incrémenté de 1 puis sa valeur est attribuée à "a"

Les opérateurs logiques (sur entiers uniquement)

Symbole	Opération bit à bit
~	complémentation (non logique)
&	ET logique
|	OU logique
^	OU exclusif
<<n	décalage à gauche de n bits
>>n	décalage à droite de n bits

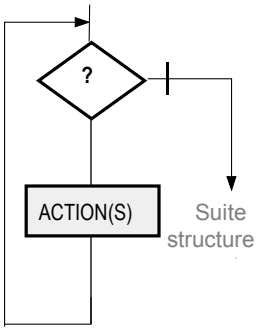
Les opérateurs de comparaison

Symbole	Fonction
= =	égale à
!=	différent de
>	supérieur à
<	inférieur à
>=	supérieur ou égal à
<=	inférieur ou égal à

Ces opérateurs renvoient la valeur "1" si la condition vérifiée est vraie.

7- Les boucles et branchements multiples

La boucle : while (tant que ... faire ...)

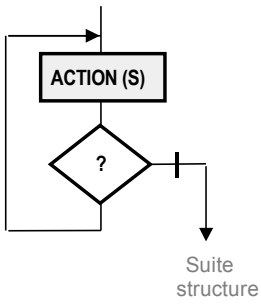



```
while (cond = = 1)
    {
    actions;
    }
```

Exemple de boucle infinie :

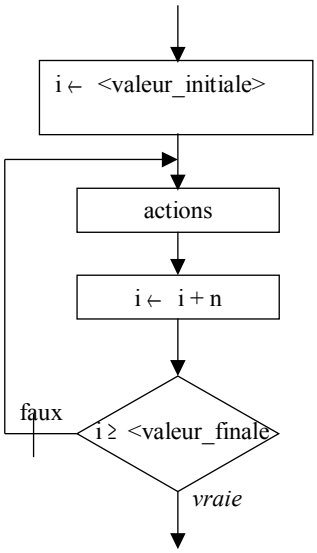
```
while (1)
{
actions;
}
```

La boucle : do ... while (faire ...tant que ...) ou (répéter jusqu'à)




```
do
{
 actions ;
}
while (cond = =1);
```

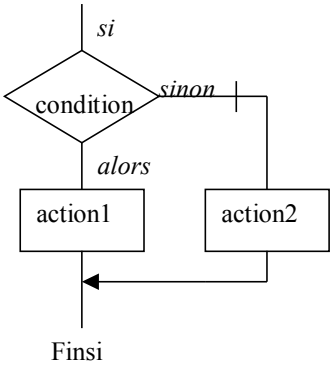
La boucle : for




```
for (i=0 ;i<5 ;i++)
{
 actions ;
}
```

remarque : la séquence présente entre accolades s'exécutera cinq fois (i variant de 0 à 4).

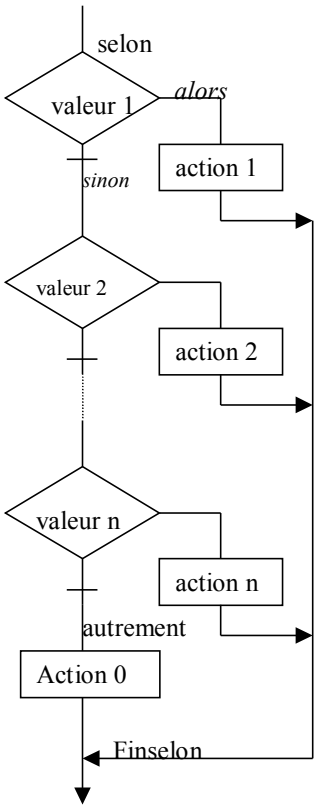
La structure : if ... else (si condition vrai faire ... sinon faire ...)




```
if (cond=1)
    {
    action1;
    }
else
    {
    action2;
    }
```

remarque : il est possible d'utiliser aussi le mot clef **else if** qui permet au sein d'une structure alternative de multiplier les choix (>2).

La structure : switch ... case




```
switch (PINA)
  {
  case (0b01111111) :
     {
     PORTB = 0b00000000 ;        /* PORTB = 0x00 si PINA = 0xEF */
     break;                      /* L'instruction break permet de sortir du switch dans le cas où la condition case est validée */
     }
  case (0b10111111) :
     {
     PORTB = 0b00001111 ;
     }
  case (0b11011111) :
     {
     PORTB = 0b11110000 ;        /* PORTB = 0x0F si PINA = 0xBF */
     }
  default : PORTB =0b11111111 ;  /* L'instruction default permet de traiter les cas non vus par les cases */
  }
```

8- DIRECTIVES DE PRÉ-COMPILATION

#include <90S8515.h>	Cette directive permet d'inclure au programme en cours le fichier de configuration des registres du composant AT90S8515. Dans le cas ou un composant de type AT90S8535 est employé il suffit d'employer le fichier 90S8535.h . Ces fichiers se trouvent dans le répertoire cavr\inc\
#define	Cette directive permet de créer des "macros instructions". Exemples : #define ALFA 0xff : Cette commande permet l'attribution de la valeur "0xff" au symbole "ALFA". Lors de la pré-compilation tous les symboles ALPHA rencontrés seront remplacés par la valeur 0xff