Décomposition d'un programme C

Un **programme C** est composé de plusieurs **blocs**. Chacun d'entre eux ayant un rôle particulier dans une ou plusieurs phases de la compilation ou de l'exécution:

Exemple :

Inclusions

// indique le nom des fichiers à insérer dans le fichier source
// exemple : #include <stdio.h> (voir TP)

#définitions de macros

// permet de définir un ensemble de macro variables et macro constantes
// voir le document

// **http://eric.bachard.free.fr/UTBM_LO20_A07/Documentation/gcc/preprocesseur_et_macros.pdf**

Définitions des variables globales // les variables déclarées dans ce bloc seront accessibles par toutes
// les fonctions du programme, y compris main()

Définitions des prototypes de fonctions // définitions qui donnent les règles d'usage des fonctions
// utilisées dans la suite du programme : nombre et types des arguments
// ainsi que le type de la valeur retournée

// main () est la fonction principale, qui sera reconnue par le linker. Cette fonction doit exister et être
// unique. Elle est la seule fonction pouvant interagir avec le système d'exploitation
// Un binaire exécutable qui n'a pas de main() est une bibliothèque (library en anglais)

```
int main( ....)
  {
        définition des variables locales à main()   // ces variables ne sont pas accessible de puis
                                                                  // l'extérieur de la fonction
        instructions      // toute expression valide en langage C devient une isntruction dès qu'elle est
                          // suivie d'un point virgule
  }

fonction1(paramètres)
  {
        définition des variables locales à fonction1
        instructions
  }

fonction2(parametres)
  {
        définition des variables locales à fonction2
        instructions
  }
```

... etc

Note : aucune structure de contrôle, ni fonction complexe n'est abordée dans cette partie

Les accolades marquent le début et la fin d'un bloc.

Mots réservés du langage C

Les mots réservés ou mots clés servent à la sémantique du langage. Ils ne peuvent être utilisés comme identificateurs ou variables, et sont utilisés en tant que :

- spécificateur de type d'objet
- spécificateur de classe d'allocation
- opérateur symbolique
- instruction de contrôle
- étiquette de contrôle

Mots réservés du langage

Mots réservés du Langage C		
auto	extern	short
break	float	sizeof
case	for	static
char	goto	struct
continue	if	switch
default	int	typedef
do	long	union
double	register	unsigned
else	return	while
const	signed	volatile
enum	void	asm

Notation : en jaune, mots réservés appartenant à la définition du langage C par Kernigan & Ritchie
En bleu, celle ajoutée par la norme ANSI

Mot clés en fonction de leur utilisation :

Type	**Classe**	**Instruction**	**Operateur**	**Etiquette**
int	auto	if	sizeof	case
char	extern	else		default
short	static	while		
long	register	do		
unsigned	typedef	for		
float		switch		
double		break		
struct		continue		
union		goto		
enum		return		

Note : il n'y a aucun mot clé du type READ, WRITE, PRINT. Cela signifie que le langage C ne dispose d'aucune

instruction d'entrée sortie par défaut -> il faudra écrire la fonction !!

Les variables simples

Les variables sont avec les opérateurs les deux éléments de base du langage C. Leur combinaison permet de former des expressions et des instructions, éléments constitutifs du langage.

On distingue deux familles de variables :

- les variables simples (scalaires)
- les variables composées (étudiées dans un autre cours)

Syntaxe de déclaration d'une variable :

Permanence classe_de_stockage modifieur type identificateur = valeur

Critère de permanence : indique si la variable est susceptible de changer indépendamment du programme ou si la variable peut être modifée par le programme. Ce critère défini dans la norme ANSI n'est pas pris en compte par tous les compilateurs.

Critères de permanence possibles :

- const : sert à informer le compilateur qu'une variable ne peut être modifiée ; le code peut alors être optimisé en conséquence. En particulier, les variables constantes sont placées dans une section particulières du programme après compilation.
- Volatile : sert à informer le compilateur qu'une variable peut être modifiée par un agent extérieur au programme (souvent un dispositif hardware)

Classe de stockage d'une variable : indique la façon et l'endroit où sera stockée la variable. Les possibilités sont :
- **auto**
- **static**
- **extern**
- **register**

Modifieur (facultatif) : destiné à agir sur le type de données. Modifieurs possibles :
- **signed**
- **unsigned**
- **short**
- **long**

Type de données : c'est un des types reconnus par le langage C. Il y a plusieurs types possibles.

- types de base : **char, int, float, double, void**
- **tableaux et pointeurs**
- les types définis par l'utilisateur : **struct, union, enum, typedef**

Identificateur : c'est le nom de la variable déclarée.

Le terme identificateur désigne le nom des variables, des fonctions, des macro instructions, des types, etc. Les identificateurs obéissent aux règles de construction suivantes:

un identificateur est formé d'une suite de lettres (A à Z, a à z), de chiffres (0 à 9) ou de caractère(s) de soulignage;
Le premier caractère d'un identificateur ne peut pas être un chiffre;
Les identificateurs sont sensibles à la casse;
il n'y a pas de limite à la longueur d'un identificateur. Toutefois, seuls les 31 premiers caractères sont en général significatifs.

Remarques importantes concernant le nom des variables :

- Les noms sont composés de chiffres et lettres
- Le premier caractère doit être une lettre
- Le caractère de soulignement " _ "appelé underscore compte comme une lettre. Son emploi est déconseillé pour le début du nom d'une variable, car cet usage est réservé aux bibliothèques qui font un usage important de cette convention de nommage.
- Par contre, dans un nom composé, son emploi est très utilisé. Exemple : ne_xml_valid.
- L'usage le plus répandu consiste à utiliser les minuscules pour les noms de variables, et les majuscules pour les constantes symboliques.
- Longueur d'un nom de variable : le compilateur tient compte des 31 premiers caractères des noms internes au minimum.

On ne peut pas utiliser les mot clés comme noms de variable.

Il est essentiel de bien nommer les variables :

- nom court pour les variables locales (indices dans une boucle ..etc)
- nom descriptif pour les variables externes (facilitant la comprehension du code en même temps)

Les types de données et leurs tailles :

char : un seul octet, dont la valeur représente un caractère du clavier de la machine utilisée, par exemple.

int : nombre entier : le nombre d'octets utilisé pour le représenter, dépend de la machine (2 ou 4).

Il est vivement recommandé de vérifier la taille d'un entier. (avec sizeof() par exemple)

Représentation des grandeurs dans le systèmes numériques

Remarque : représentation de -8 à +7 en binaire

Valeur en décimal	$b_3 \cdot 2^3$	$b_2 \cdot 2^2$	$b_1 \cdot 2^1$	$b_0 \cdot 2^0$
0	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**
2	**0**	**0**	**1**	**0**
3	**0**	**0**	**1**	**1**
4	**0**	**1**	**0**	**0**
5	**0**	**1**	**0**	**1**
6	**0**	**1**	**1**	**0**
7	**0**	**1**	**1**	**1**
-8	**1**	**0**	**0**	**0**
-7	**1**	**0**	**0**	**1**
-6	**1**	**0**	**1**	**0**
-5	**1**	**0**	**1**	**1**
-4	**1**	**1**	**0**	**0**
-3	**1**	**1**	**0**	**1**
-2	**1**	**1**	**1**	**0**
-1	**1**	**1**	**1**	**1**

Intérêt de ce codage : il donne toujours un résultat correct par addition binaire, quel que soit le signe des grandeurs.

Exemple : 6 + (-5) => **$6_{10} = 0110_2$, $-5_{10} = 1011$ -> 0100 + 1011 = 0001**

Sur 16 bits :

Décimal	-128 ..... +127	0	+1 ..... +127
Binaire en complément à 2	$80 .... à ..... $FF	$00	$01 ..... $7F

Important : discontinuité des valeurs binaires, continuité des valeurs numériques

Représentation en complément à deux décalé

Permet d'obtenir la continuité du poids décimal global de la représentation binaire. Par exemple, pour une représentation sur 8 bits, on additionne $80 au résultat obtenu par représentation en complément à 2

Décimal	-128 ....-1	0	+1 ..... +127
Binaire en complément à 2	$80 ..... $FF	$00	$01 ..... $7F
Binaire en complément à 2 décalé	$00 ..... $7F	$80	$81 ..... $FF

Exemple d'application : convertisseur analogique numérique, à 8 bits, de résolution.

En entrée : mot de 8 bits, en sortie tension pouvant variant entre -Vmax et +Vmax

Représentation du type réel [FIXME]

Les types

En C il faut déclarer les variables avant de les utiliser, en général au début de la fonction, avant la première instruction exécutable

Note: ce n'est plus vrai avec la norme C99 : on peut déclarer les variables n'importe où dans la fonction.

Commentaires : placés entre /* et */

Intérêt : même sur plusieurs lignes. On ne peut pas imbriquer deux commentaires (essayer de voir pourquoi).

Le type *char* :

Utilisé pour déclarer des variables prenant par exemple pour valeur celle d'un caractère, généralement codée sur 8bits. Ce type permet le stockage de tous les caractères de la norme ASCII, mais aussi les caractères ASCII étendus de la norme IBM PC (allant de 127 à 255).

Valeurs possibles :
unsigned char -> 0 à 255
char -> -128 +127

Le type *int* (pour *integer*, qui signifie entier) :

Dépend de la machine utilisée : un entier peut être codé sur 16 ou 32 bits, en fonction de la machine utilisée.
- sur 16 bits : désigne une valeur comprise entre -32768 et +32767
- sur 32 bits : désigne une valeur comprise entre -2 147 483 648 et 2 147 483 647

Le type *float* (réel) :

Le type **float** est utilisé pour déclarer des variables prenant pour valeur un nombre réel en virgule flottante positif ou négatif. La valeur de la variable occupe **4 octets en mémoire.**

Valeurs possibles comprises entre -10^{-38} et 10^{-38} avec un pas minimal de 10^{-38} et une précision de 6 décimales

Le type *double* :

réel occupant 8 octets en mémoire.

Le type *long double* :

occupe 10 octets en mémoire, 18 chiffres significatifs

Déclaration :

En algorithme :

```
réel  taux_de_TVA
réel  note_moyenne
réel distance
```

En C **:**

```
float taux_de_TVA;
float moyenne;
double distance;
```

Type *void* :

Le type void est principalement utilisé dans les déclarations de fonctions. Comme type de fonction, il précise qu'aucune valeur n'est renvoyée par la fonction (comme une procédure en Pascal).
Comme unique type de paramètre, il indique que la fonction ne possède aucun paramètre.

Les modifieurs

Le modifieur *signed*

Le modifieur ***signed*** peut être employé sur les types ***char*** et ***int***. Il a pour efet de centtrer autour de 0 les valeurs possibles (donc de diviser par 2 la plage possible en valeur absolue).

Type	**Minima**	**Maxima**
signed char	**-128**	**+127**
signed int	**-32768**	**+32767**
signed long int	**-2 147 483 648**	**+2 147 483 647**

Le modifieur *unsigned*

Le modifieuur ***unsigned*** peut être utilisé sur les types ***char*** et ***int.*** Il a poour résultat de rendre la borne inférieure des valeurs possibles égale à 0.

Type	**Minima**	**Maxima**
unsigned char	**0**	**255**
unsigned int	**0**	**65535**
unsigned long int	**0**	**4 294 967 295**

Une des utilisations le s plus courantes de ce modifieur est la définition d'un type BYTE :

```
#define BYTE unsigned char
```

Le modifieur *short* :

Le modifieur *short* peut être utilisé avec les types *char* et *int*. Il a pour effet de réduire l'intervalle des valeurs du type.

Type	**Minima**	**Maxima**
short char	**0**	**127**
short int	**-32768**	**32767**

Remarques :
- **short est équivalent à short int**
- **une variable de type short occupe toujours 2 octets en mémoire**
- **short est considéré comme un type de données simple. Unsigned et signed peuvent donc être appliqués a short**

Type	Minima	Maxima
unsigned short int	-32768	+32767
Signed short int	0	+65535

Le modifieur long :

Le modifieur *long* peut être employé comme type de données simple. u*nsigned* et *signed* peuvent donc lui être appliqué.

Type	**Minima**	**Maxima**
signed long int	**-2 147 483 648**	**+2 147 483 647**
unsigned long int	**0**	**+4 294 967 295**

Constantes caractères :

Les constantes caractères définissent toutes des données formées d'un seul caractère.

Syntaxe : 'caractere' /* malgré la présentation, il s'agit d'un seul caractère à la fois */

caractere est un caractère ASCII imprimable, à l'exception de ' et de \ qui devront être "échappés".

Exemples :

'a' lettre a minuscule, code décimal 64
'@' caractère a commercial, code décimal 97
'V' lettre v majuscule, code décimal 86

Constantes chaines de caractère

Syntaxe : "suite de caractères" Type : toutes les chaines de caractères sont du t**ype tableau de *char***

Ici, cette chaine de caractères contient les éléments suivants :

s, u, i, t, e, "caractère espace" , d, e, "caractère espace", c, a , r, a, c ,t , è , r, e, s + le caractère **NUL (code ASCII 0)** noté aussi **\0**

Cette chaine de caractères occupe donc en mémoire : 19 caractères + \0 en tout, c'est à dire 20 caractères