Polycopiés

Cours Informatique embarquée

GEII Toulouse 2020 - 2021

A. Nketsa

CHAP 5 Langage C (Rappel partie 1)

Langage C (Rappel partie 1)

- **Variables**
- **Le pointeur**
- **Opérateurs**
- **Instructions**

Langage C (Rappel partie 2)

- **Fonctions**
- **Notion de composant logiciel**
- **Organigramme structuré basé composants**

Exemple de traduction d'organigrammes en langage C

Langage C (partie 3)

- **Fonctions d'entrées-sorties standard en langage C**
- **Représentation sous forme de composants logiciels**

CHAP 5 Langage C

I- (Partie 1 Rappel)

Remarques importantes

I-En S1, vous avez programmé en langage C dans un environnement de développement en C++.

Cela ne veut pas dire que vous savez programmer en C++, parce que en C++ on peut manipuler les objets et les classes que vous n'avez pas vus.

II-En S2, nous allons travailler en langage C pour l'informatique embarquée dans un environnement en C.

III- Conclusion

En S2, nous allons utiliser pratiquement les mêmes instructions vues en programmation qu'en S1

3 points vont un peu changer :

a) Les instructions d'entrées sorties standard

	En C++ **C'est le compilateur qui fait presque tout pour vous pour afficher**	**En C standard** **C'est vous qui devez dire au compilateur ce que vous souhaitez afficher**
Entrée standard	**Cin**	**Lecture d'un caractère** getkey(void) getchar(void) **Lecture d'une chaîne de caractères** gets(char *p_ch, ui nb_car)
Sortie standard	**Cout**	**Ecriture d'un caractère** putchar(char code_ascii) **Ecriture d'une chaîne de caractères** puts(char *p_ch) **Ecriture formatée** printf("msg + balises", parameters)

b) Les pointeurs

Vous les avez vus en fin de S1.

Comme c'est un **point très important en langage C**, nous y reviendrons

c) Le passage des paramètres d'entrées-sorties des fonctions

En S1, vous avez vu le passage des paramètres d'entrées-sorties des fonctions par référence. C'est-à-dire, pour simplifier, que c'est le compilateur qui fait la correspondance entre les paramètres formels et ceux effectifs.

En langage C dans un environnement de développement en langage C, c'est le programmeur qui est reponsable de ce passafe de paramètres.

Nous allons y consacrer un chapitre

5-1 Langage C

1- Les Variables (Type – Taille – Occupation mémoire - Déclaration)

Définition

Une variable est un ensemble de cases mémoire (généralement) consécutives.
A cet ensemble de cases, on associe :

- **un nom**
- **une adresse (c'est l'adresse de la première case de l'ensemble)**
- **un contenu**

Donc déclarer une variable c'est :

- Donner un nom

- Indiquer la taille de la variable et l'interprétation

- Associer l'adresse de la variable à son nom

Obtention du contenu d'une variable

Pour obtenir le contenu d'une variable,

il suffit d'indiquer le **nom de la variable**

Obtention de l'adresse d'une variable

Pour obtenir l'adresse d'une variable,

il suffit d'utiliser l'opérateur **&**

Exemple : **&nom_variable** donne l'adresse de la variable appelée **nom_variable**

Taille des variables sur microcontrôleur courant

Nom du type	commentaire	taille	intervalle
char	**Nombre signé**	**8bits (1octet = case mémoire)**	**-128 à +127**
unsigned char	**Nombre non signé**	**8bits**	**0 à 255**
int	**Nombre signé**	**16bits**	**-32768 à +32767**
unsigned int	**Nombre non signé**	**16bits**	**0 à 65535**
long	**Nombre signé**	**32bits**	**-2147483648 à -+2147483647**
unsigned long		**32bits**	**0 à 4294967295 =(2^{32} -1)**
float		**32bits**	**≈ -10^{-38} à $+10^{38}$**

2- Variables de types scalaires (simples)

1- Déclaration :

Syntaxe générale : type nom_variable **= valeur_initiale (optionnelle);**

2- Organisation mémoire des variables et stockage

Base de stockage = **octet**

Pour une lecture simple, la notation en hexadécimale des valeurs est conseillée

Exemple

le mot de 16bits 0x5421 => poids fort = 0x54 poids faible = 0x21

Stockage de mots de plus de 8bits(octet) dans la mémoire :

a) Mode **little endian** : poids faible puis poids fort dans l'ordre croissant des adresses

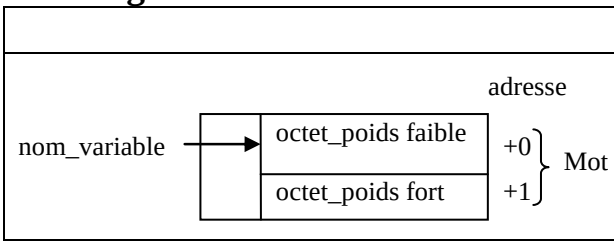
adresse	Type	Taille	Nom variable	contenu	application
+0	int ou	2	Nom_variable	Poids faible	
+1	unsigned int			Poids fort	

Exemple

unsigned int mot_16bits = 0x5421;

adresse	Type	Taille	contenu	contenu	application
+0	int ou	2	mot_16bits	Poids faible	0x21
+1	unsigned int			Poids fort	0x54

Stockage de mot de 16bits : little Endian

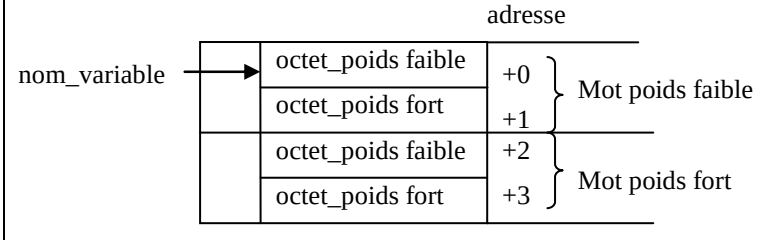


Exemple mot = 0x55AA

unsigned int mot =0x55AA

Nom variable	adresse	contenu
mot	0x51	0xAA
	0x52	0x55

Stockage de mot de 32bits : little Endian



Exemple mot_32bits=0x00FF55AA

float reel = 0x00FF55AA

Nom variable	adresse	contenu	
reel	0x100	0xAA	Mot poids faible
	0x101	0x55	
	0x102	0xFF	Mot poids fort
	0x103	0x00	

b) Mode **big endian** : poids fort puis poids faible dans l'ordre croissant des adresses

adresse	Type	Taille	Nom variable	contenu	application
+0	int ou	2	Nom_variable	Poids faible	
+1	unsigned int			Poids fort	

Exemple

unsigned int mot_16bits = 0x5421;

adresse	Type	Taille	contenu	contenu	application
+0	int ou	2	mot_16bits	Poids fort	0x54
+1	unsigned int			Poids faible	0x21

Exemples de déclaration des variables simples (scalaires)

```
char            caractere, octet_sg;  // taille 1 octet      valeur entre -128 et +127
unsigned char   octet_ns;             // taille 1 octet      valeur entre 0 et 255
int             mot_sg_16bits         // taille 16bits = 2 octets
                                      //                     valeur entre -32768 et +32767
unsigned int        mot_ns_16bits     // taille 16bits = 2 octets
                                      //                     valeur entre 0 et +65537
float           nb_reel;              // taille 32bits = 4 octets = 2 mots de 16bits
                                      //                     Valeur ≈ -10^-38  à +10^+38
```

3- Variables de type structuré

1- Définition et types structurés

Une variable de type structuré est une variable qui regroupe sous une appelation plusieurs variables :

- de même type, c'est un tableau
- de types différents, c'est une structure

Nous ne traiterons que de la variable structurée de type tableau

2- Tableau à une dimension

Déclaration:

Sans valeur initiale :

type nom_tableau[nombre_eléments];

Avec valeur initilae :

type nom_tableau[nombre_eléments] =(optionnelle) {valeur initiales, };

Exemple

Sans valeur initiale

unsigned char tab_uc[3];

Organisation du tableau

tab_uc	tab_uc[0]	
	tab_uc[1]	
	tab_uc[2]	

Avec valeur initiale

unsigned char tab_uc[3] = {0x20, 0x41, 0x45};

Organisation du tableau

tab_uc	0x20	tab_uc[0]
	0x41	tab_uc[1]
	0x45	tab_uc[2]

3- Tableau à deux dimensions

Déclaration :

Sans valeur initiale :

type nom_tableau[nombre_ligne][nombre_colonne];

Avec valeur initiale :

type nom_tableau[nombre_ligne][nombre_colonne] ={ {valeur initiales}, };

Exemple

Sans valeur initiale

unsigned int tab_ui[2][2];

Organisation du tableau

tab_ui	tab_ui[0]	tab_ui[0][0]	Ligne 0
		tab_ui[0][1]	
	tab_ui[1]	tab_ui[0][0]	Ligne 1
		tab_ui[0][1]	

Avec valeur initiale

int tabi[3][2] = {{25, -30},{46,00},{100,1}}

	colonne0	colonne1
ligne0	tabi[0][0]	tabi[0][1]
ligne1	tabi[1][0]	tabi[1][1]
ligne2	tabi[2][0]	tabi[2][1]

Organisation tableau

tabi	tabi[0]	tabi[0][0]	25	Ligne0
		tabi[0][1]	-30	
	tabi[1]	tabi[1][0]	46	Ligne1
		tabi[1][1]	00	
	tabi[2]	tabi[2][0]	100	Ligne2
		tabi[2][1]	1	

3- Accès aux éléments d'un tableau à une dimension

Désignation de tout le tableau:

nom du tableau

Désignation d'une case du tableau :

nom_tableau[indice] avec 0 ≤ indice < nombre d'éléments du tableau

4- Accès aux éléments d'un tableau à deux dimensions

Désignation de tout le tableau:

nom du tableau

Désignation d'une case du tableau :

nom_tableau[ligne][colonne] avec 0 ≤ ligne < nombre de lignes du tableau
avec 0 ≤ colonne < nombre de colonnes du tableau

Désignation d'une ligne su tableau:

nom_tableau[ligne] avec 0 ≤ ligne < nombre de lignes du tableau

Remarque importante :

- Le nom du tableau représente **l'adresse du tableau**
- **L'adresse du tableau est l'adresse de la première case du tableau**
  - pour un tableau à 1 dimension
    nom_tableau ≡ &nom_tableau[0]

  - pour un tableau à 2 dimensions
    nom_tableau ≡ &nom_tableau[0][0]
    ≡ nom_tableau[0] pour la première ligne

Cas particuliers des chaînes de caractères

Définition :

Une chaîne de caractères est un tableau à 1 dimension qui :

- contient des codes ascii,
- et se termine par le **séparateur 0x00**

Notation

- le code ascii d'un symbole est noté entre apostrophes : 'symbole'
  Exemple **'A'** codes ascii de A

- une chaîne de caractères est notée entre guillemets : "suite de symboles"
  Exemple : **"ABCD"**

  char chaine[6] = **"ABCD";**

chaine	chaine[0]	chaine[1]	chaine[2]	chaine[3]	chaine[4]	chaine[5]
	'A'	'B'	'C'	'D'	**0x00**	Non défini
					Fin de la chaine	

1-3 Type pointeur

a) Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable.

Autrement dit : **un pointeur** contient une **valeur** qui ne peut être interprétée que comme une **adresse**.

Plus simplement, on dit qu'un pointeur contient l'adresse d'une autre variable que l'on appelle variable pointée.

b) Déclaration d'un pointeur

Sans valeur initiale :

```
type*   nom_pointeur;        // notée plus couramment   type   *nom_pointeur;
```

Avec valeur initiale :

```
type*   nom_pointeur=adresse;
```

c) Compréhension

1) le pointeur **pointe** sur une variable de **même type que celui de la déclaration de la variable**.

2) nom_pointeur est la variable **pointeur** qui contient l'**adresse** de la variable pointée

3) *nom_pointeur est le **contenu** de la **variable pointée**

4) nous conseillons de noter le pointeur sous la forme p_nom pour le différencier des variables standard

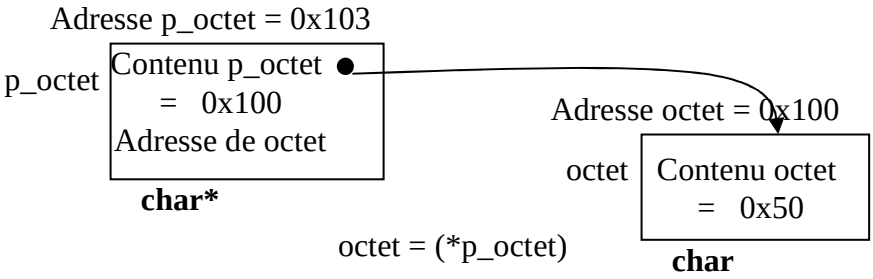
5) **Exemple :**

```
unsigned char octet = 0x50;
unsigned char *p_octet;
```

Schéma

Nom_variable	adresse	contenu	interprétation
octet	0x100	**0x50**	Valeurs des variables
	0x101	78	
	0x102	0x54	
p_octet	0x103	**0x100**	Adresse de la variable pointée
	0x104		

5-1) Schéma



5-2) le contenu de p_octet est **0x100 = adresse variable pointée**

5-3) le contenu de ***p_octet est le contenu de la variable** qui se trouve à l'adresse **0x100** donc le contenu de la variable octet

5-4) **Conclusion (*p_octet) = 0x50**

d) Règles de mainpulation des pointeurs

- on peut ajouter une valeur entière à un pointeur ⇒ nom_pointeur + valeur_entière
- on peut soustraire une valeu entière d'un pointeur ⇒ nom_pointeur - valeur_entière

Valeur_entière peut être :

- une constante entière
- une variable de type char, unsigned char, int ou unsigned int

En savoir plus

Pour calculer l'adresse pointée par ces deux opérations, il faut tenir compte de taille du type pointé.

adresse pointée = (nom_pointeur) ± (taille_type_pointeur * valeur_entière)

Conseil d'utilisation des pointeurs

Nous conseillons d'utiliser les parenthèses **(*nom_pointeur)** pour éviter des ambiguïtés d'interprétation dues à la priorité entre opérateurs.

Relations Pointeur – Tableau

Rappel : nous avons déjà dit que le **nom du tableau** est **l'adresse du tableau**

On peut donc considérer que le nom d'un tableau est un pointeur qui contient l'adresse fixe du tableau.

Considérons la déclaration	**on peut écrire :**	
unsigned int tab_ui[5];	p_uint = tab_ui;	// p_uint reçoit l'adresse de tab_ui
unsigned int *p_uint;	p_uint = &tab_ui[indice];	// p_uint reçoit l'adresse de tab_ui[indice]

Considérons :

```
p_uint = tab_ui;          // p_uint reçoit l'adresse de tab_ui
```

Les écritures suivantes sont indentiques

```
tab_ui[indice]   ≡   *(p_uint + indice)     // la parenthèse est obligatoire pour éviter l'ambigüité
tab_ui[2]        ≡   p_uint[2]              // sans *
p_uint[2]        ≡   *(p_uint + 2)
```

2- Les opérateurs

a) Arithmétiques de base (+, -, *, /)

Opérateur	Interprétation		Syntaxe	commentaires
+ (binaire)	addition		x + y	
- (binaire)	soustraction		x - y	
* (binaire)	Multiplication		x * y	
/ (binaire)	Division entière		x / y	Quotient de x / y
% (binaire)	modulo		x % y	Reste de la division entière x / y
+ (unaire)	Signe positif		+ x	
- (unaire)	Signe négatif		- x	
+ (unaire)	Incrémentation	après	x++	
+ (unaire)		avant	++x	
- (unaire)	Décrémentation	après	x--	
- (unaire)		avant	--x	

Hiérachie des opérateurs

Régle simple : utilisation des parenthèses pour expliciter le calcul

b) Affectation

Opérateur	Interprétation	Syntaxe	commentaires
=	Affectation simple	x = y;	
		x = formule	
(opérateur) =	Affectation composée	x operateur = y;	x = x opérateur y;

c) Opérateurs mathématiques

Les opérateurs mathématiques sont des fonctions dont on récupère les résultats.

Exemples de fonctions

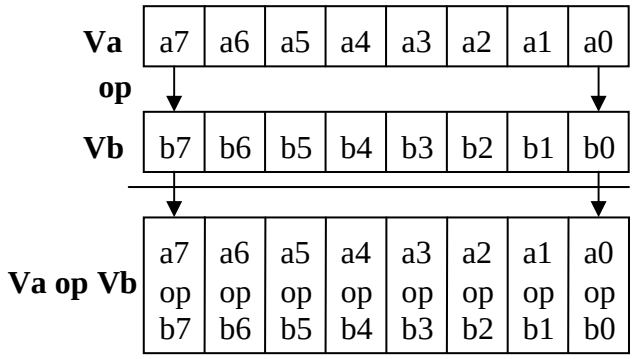
librairies	fonction	interprétation
#include <math.h>	float pow (float *x*, float *y);*	Resultat = x^y
	float sin (float *x*);	Résultat = sin (x)

d) Opérateurs logiques

Définition

Un opérateur logique permet d'effectuer une opération logique entre les bits de même rang dans les opérandes sans report.

Exemple :



Opérateurs logiques de base

Opérateur	Interprétation	Syntaxe	commentaires
~(unaire)	Pas	~X	
&(binaire)	ET	X & Y	
| (binaire)	OU	X | Y	
^ (binaire)	OU exclusif	X ^Y	

Opérateurs logiques de décalage

>> (binaire)	Décalage à droite	X >> a	Décalage à droite de a position Exemple **Va** a7 a6 a5 a4 a3 a2 a1 a0 Va >> 1 0 → 0 a7 a6 a5 a4 a3 a2 a1 →
<< (binaire)	Décalage à gauche	Y << b	Décalage à gauche de b position Exemple **Vb** b7 b6 b5 b4 b3 b2 b1 b0 Vb << 1 ← b6 b5 b4 b3 b2 b1 b0 0 ← 0

Utilisation des opérateurs logiques en informatique industrielle et/ou embarquée

Principe
Définir le masque d'isolement des bits concernés
⇒ **mettre 1 dans la position des bits concernés et 0 pour tous les autres bits**
Nous l'appellerons masque en ET que nous noterons masque_ET

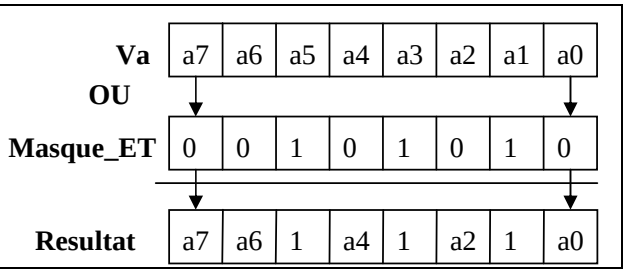
Exemple considérons une variable de 8bits, définir le masque en ET pour les bits 7, 4 et 0
masque_et = 0b10010001 = 0x91

Nous pouvons aussi définir un masque en OU que nous noterons masque_OU pour lequel on met 0 dans la position des bits concernés et les autres à 1.

a) Mise à 1 d'un ou plusieurs bits d'un mot
Effectuer un OU logique entre le mot et le masque d'isolement

Principe :
- définir le masque en ET
- faire OU logique entre le mot et le masque en ET

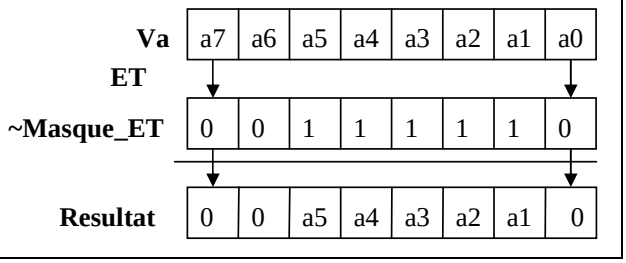
Exemple
mettre à 1 les bits 5, 3 et 1 de la variable va
- masque_ET = 0b00101010 = 0x2A
- effectuer le OU Va | masque_ET



b) Mise à 0 d'un ou plusieurs bits d'un mot
Effectuer un ET logique entre le mot et **le complément** du masque en ET

Principe :
- définir le masque en ET
- faire ET logique entre le mot et le complément du masque en ET

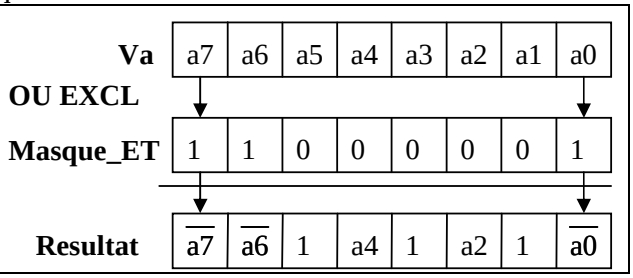
Exemple
mettre à 1 les bits 7, 6 et 0 de la variable va
- masque_ET = 0b11000001 = 0xC1
- ~masque_ET = 0b00111110 = 0x3E
- effectuer le ET Va & ~masque_ET



c) Complémentation d'un ou plusieurs bits d'un mot
Effectuer un OU EXCL logique entre le mot et le masque en ET
$S = A \oplus B = A\overline{B} + \overline{A}B$ donc $B = 0 \Rightarrow S = A$
$B = 1 \Rightarrow S = \overline{A}$

Principe :
- définir le masque en ET
- faire OU_exclusif logique entre le mot et le masque en ET

Exemple
Complémenter à 1 les bits 7, 6 et 0 de la variable va
- masque_ET = 0b11000001 = 0xC1
- effectuer le ET Va ^ masque_ET



e) Opérateurs Booléens

Définition
Les opérateurs booléens sont les opérateurs pour lesquels, le résultat produit est un booléen.

Remarque :
Un booléen est une valeur vrai ou faux

Pour une présentation simple, nous allons répartir les opérateurs booléen en deux groupes

- **Opérateurs booléens simples**
- **Opérateurs booléens composés**

Opérateurs Booléens simples

Opérateurs de comparaison
Un opérateur de comparaison permet d'effectuer une opération de soustraction suivie d'une prise de décision booléenne (Vrai – Faux). On dit aussi qu'il produit un résultat booléen.

Opérateur	Interprétation	Syntaxe	commentaires
==	égalité		
!=	différent		
<	Inférieur (plus petit)		
<=	Inférieur ou égal		
>	supérieur (plus grand)		
>=	supérieur ou égal		

Exemple

Opérateurs Booléens composés

Opérateurs booléens pour la combinaison
Un opérateur booléen permet d'effectuer des fonctions logiques entre booléen pour obtenir un résultat booléen.

Opérateur	Interprétation	Syntaxe	commentaires
!(unaire)	Pas (NON)		
&&(binaire)	ET booléen		
|| (binaire)	OU booléen		

Exemple
On considère la variable unsigned int mot1, on veut tester
a) si les bits pairs sont tous à 0

b) si les bits 15, 13, 10, 8, 4 et 2 sont respectivement à 0, 1, 0, 0 ,1 et 1

Opérateurs d'accès à la mémoire

opérateur	interprétation	syntaxe	commentaire
&	Obtention de l'adresse de	&nom_variable	Fournit l'adresse de nom_variable
*	Indirection (contenu du contenu)	*ptr	ptr pointe sur une variable ptr contient l'adresse de la variable (*ptr) contient la donnée

Autres opérateurs

opérateur	interprétation	syntaxe	commentaire
(type)	**Cast** forçage de l'interprétation	(int) var_char	Interpréte var_char comme un entier signé (int)
siezof	Taille d'une variable en nombre de bits	Sizeof(var_int)	Nombre de bits de la variable var_int
?: :	Evaluation conditionnelle	v ?:z : t	Si v ≠ 0 alors y sinon t

Instructions

Fonctions

Notion de composant logiciel

1- Affectation

L'instruction d'affection consiste à transférer une information ou le résultat d'une expression dans une variable.

dest = source; ou dest = expression;

Expression est une formule utilisant différents opérateurs

Une instruction d'affectation peut donner lieu à des conversions implicites

2- Conversions implicites

Une conversion implicite de type est une conversion effectuée automatiquement par le compilateur soit pendant les calculs soit lors de l'affectation.

Règles de conversions implicites

- **Sans perte d'information**

  Des types les moins précis vers les plus précis

  Char → int → long → float

  Exemple

  Considérons les déclarations

```
char    car;
int     v_int;
float   reel;
```

  on peut écrire sans perte de precision :

```
v_int   =  car;         // poids fort de v_int = 0x00      et poids faible de v_int = car
reel    = car;          // la partie entière de reel = car
reel    = v_int;            // la partie entière de reel = v_int
```

- **Avec perte possible d'information**

  Des types les plus précis vers les moins précis

  float → long → int → char

  Exemple

  Considérons les déclarations

```
char    car;
int     v_int;
float   reel;
```

  on peut écrire sans perte de precision :

```
car     = V_int;        //car = poids faible de v_int           donc le poids fort est perdu
car     = reel;         // la partie fractionnaire perdue   et  |reel| doit être < 256
v_int   = reel;         // la partie fractionnaire perdue   et  |reel| doit être < 65536
```

- **pour les arguments d'une fonction, les conversions implicites sont automatiques**

3- Conversions explicites (cast)

Objectif :
Une conversion explicite a pour objectif de forcer le type d'une variable pour une untilisation donnée.

Syntaxe
(type souhaitée) nom_variable

Exemple

Considérons les déclarations

```
char    car;
int     v_int;
float   reel;
```

on peut écrire sans perte de precision :

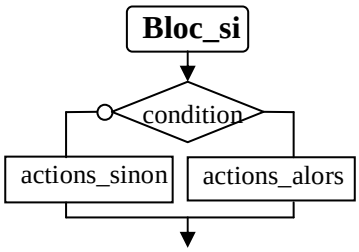
```
(int) car       // considérer car comme un int au lieu de un char
(int) reel      // considérer reel comme un int au lieu de un float
(float) v_int   // considérer v_int comme un float au lieu de un int
```

4- Instructions de structure de contrôle

Remarque importante :

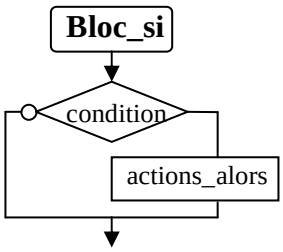
Dans les instructions de contrôle, condition est expression booléenne.

a) Choix simple (Bloc_si)



Traduction en C

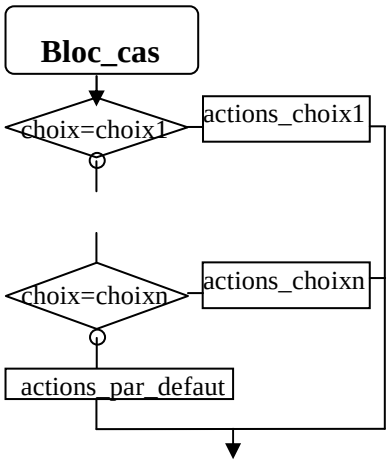
```
if (condition)
 {
   actions_alors
 }
else
 {
   actions_sinon
 }
```



Traduction en C

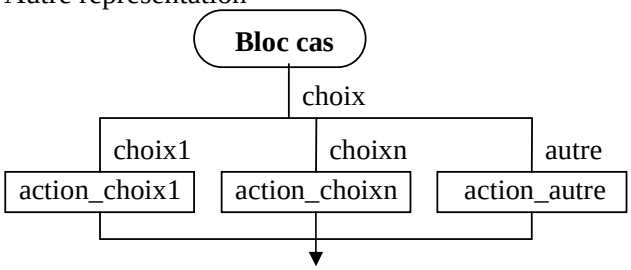
```
if (condition)
 {
   actions_alors
 }
```

b) Choix multiple




```
switch ( choix)
 { case choix1  : action_choix1
                  break ;
   case choix1  : action_choix1
                  break ;
   case choix2  : action_choix2
                  break ;


   case choixn  : action_choixn
                  break ;
   default      : action_defaut
}
```

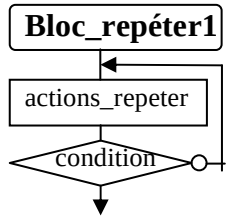
Autre représentation



c) Boucles

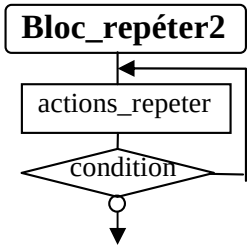
c-1 Repeter

Repeter jusqu'à condition




```
do
 {
   action_repeter
 }
while ( !(condition)) ;
```

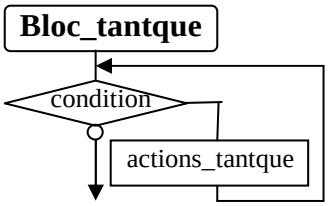
Repeter tantque condition




```
do
 {
   action_repeter
 }
while ( (condition)) ;
```

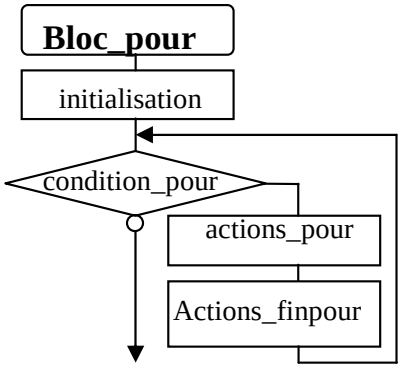
c-2 Tant que et Pour

Tant que




```
while (condition) ;
 {
   action_tant_que
  }
```

Pour



Remarque :

La structure de contrôle pour est un cas particulier de celle de tant que.

```
for (initialisation; condition_pour; actions_fin_pour)
 {
   Actions_pour
 }
```

5- Les Fonctions en langage C

a- Définition

Une fonction est une séquence d'instructions que l'on peut appeler autant de fois que l'on veut pour son exécution.
La fonction est identifiée par un nom

Pour lui permettre d'effectuer différents traitements, on lui associe des paramètres :

- dit formels à la construction
- dit effectifs à l'utilisation.

La fonction peut retourner ou pas une valeur.

Une fonction est une entité qui peut être autonome.
C'est un élément important de la structuration d'un programme.

b- Paramètres et Notion d'entrée, de sortie et d'entrée-sortie

Les paramètres d'une fonction peuvent être classés en 3 groupes que nous ramènerons à groupes

- Groupe 1 : le paramètre n'est pas modifié pour le programme qui appelle la fonction. On dit que c'est une entrée.
- Groupe 2 : le paramètre est modifié pour le programme qui appelle la fonction. On dit que c'est une sortie.
- Groupe 3 : le paramètre est utilisé dans la fonction comme une entrée et peut être modifié pour le programme qui appelle la fonction. On dit que c'est une entrée-sortie.

Remarque :

- la sortie et l'entrée-sortie peuvent être regroupées.
- Ce paramètre doit utiliser le pointeur pour indiquer l'adresse de la variable que le programme appelant veut faire modifier.

c- Notion de paramètres formels

Un paramètre formel est une variable qui apparaît dans l'entête de la fonction.
Cette variable peut être utilisée dans la fonction comme toute autre variable.
Si le paramètre est de type entrée alors la variable associée est une variable locale

Déclaration d'un paramètre formel d'entrée
Type nom_paramètre_entree

Déclaration d'un paramètre formel d'entrée-sortie ou de sortie
Type* nom_paramètre_entree_sortie

d- Variable locale

Une variable locale est une variable déclarée dans la fonction. Elle n'est vue que dans la fonction même si elle a le même nom qu'une variable globale.

Nous conseillons cependant pour des raisons de lisibilité de ne pas donner le même nom à des variables locales et globales.

e- Variables globales

Une variable globale est une variable déclarée en dehors des fonctions et du main. Elle peut être vue par toutes les fonctions et le main si elle est déclarée avant les fonctions

f- Structure d'une fonction

Définition d'une fonction (création d'une fonction)

La définition d'une fonction correspond à la création de la fonction.

fonction sans un retour

```
void      nom_fonction (déclaration des paramètres formels)
{  // déclaration des variables locales

        // corps de la fonction
}
```

fonction avec un retour

```
type_retour      nom_fonction (déclaration des paramètres formels)
{  // déclaration des variables locales

        // corps de la fonction
     return(valeur);
}
```

Dans ce cas, on doit trouver return dans la fonction.
Nous conseillons de n'avoir qu'un seul **return**

g) Utilisation d'une fonction

1) Définition
L'utilisation de la fonction consiste à appeler la fonction en remplaçant les paramètres formels par les paramètres effectifs.

2) Notion de paramètres effectifs
Un paramètre effectif est la variable ou la constante qui remplace le paramètre formel.

3) Passage de paramètres
Le remplacement du paramètre formel par celui le paramètre effectif est appelé **passage de paramètres**
Ce passage doit respecter certaines règles :

- l'ordre des paramètres
- le type des paramètres formels et effectifs

4) Syntaxe d'utilisation
Deux cas :

Pas de retour, l'appel sera

```
nom_fonction_sans_retour(passage de paramètres);
```

Avec un retour

```
resultat = nom_fonction_avec_retour(passage de paramètres);
```

6- Composant logiciel en langage C

1- Définition

Un composant en langage C est une fonction avec valeur de retour ou pas.
Il comporte deux parties :

- la vue externe appelée aussi interface, ce sont les entrées-sorties du composant. En d'autres termes, ce sont les paramètres formels de la fonction et la valeur de retour éventuelle.
- la vue interne, c'est le comportement de la fonction. En d'autres termes, c'est la suite d'instructions qui décrit ce que fait la fonction.

2- Représentation

Nous allons représenter les fonctions comme des composants avec une vue externe que nous allons utiliser dans les organigrammes

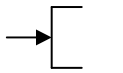
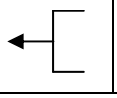
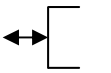
a) Représentation graphique

Dans cette représentation,

☞ **chaque paramètre formel possède un nom formel associé à un type.**

c pour char,	uc	pour unsigned char	i pour int,	ui	pour unsigned int
f pour float	d	pour double	l pour long	et ul	pour unsigned long

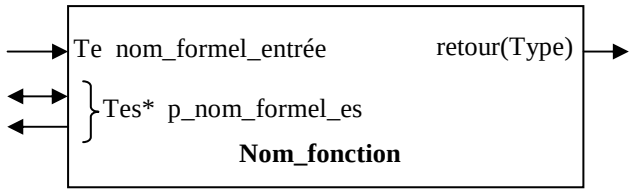
☞ **chaque paramètre formel possède une flèche pour indiquer le sens du paramètre**

Paramètre formel			
entrée	sortie	Entrée-sortie	retour
			

Remarque:

Le paramètre entrée-sortie de type pointeur s'écrit comme un paramètre de sortie seule de type pointeur ou type tableau.

Schéma général de la représentation



Prototype :
Type nom_fonction (Te nom_formel_entree,
Tes* p_nom_formel_es)

Te = type du paramètre d'entrée

Tes = type du paramètre de sortie ou d'entrée-sortie

Remarques importantes

a) Nous avons mis à gauche les paramètres formels en entrée, en sortie et entrée-sortie puis à droite le retour de la fonction

a) Il est conseillé d'éviter l'utilisation des variables globales.

b) Pour des raisons de lisibilité, nous proposons de faire précéder par p_ le nom des paramètres formels en sortie ou en entrée-sortie. Ceci permet de voir directement que le paramètre formel est un pointeur.

Littérale : fonction prototype	type Nom_fonction (type nom_formel_entree, type ***p_**nom_formel_es)

3- Passage de paramètres

Le passage de paramètres consiste à connecter des paramètres effectifs aux paramètres formels et en respectant l'ordre de remplacement.

Un paramètre formel en entrée peut être connecté à un paramètre effectif qui peut être :

- une constante notée en langage C valeur
- un contenu d'une variable noté en langage C nom_variable

Un paramètre formel pointeur peut être connecté à un paramètre effectif qui peut être :

- l'adresse d'une variable notée en langage C &nom_variable ou nom_tableau ou &nom_structure
- le contenu d'un pointeur (pointant le même type) noté nom_pointeur

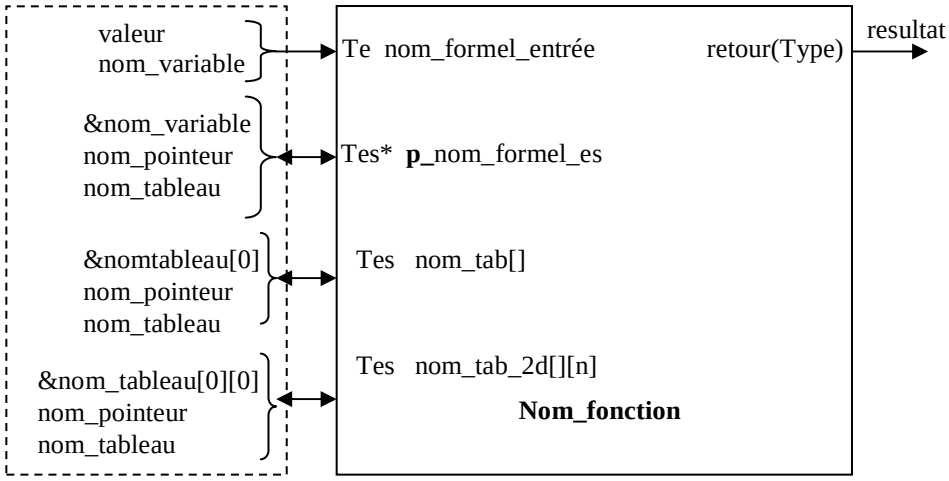
Schéma de principe du passage de paramètres

Le schéma consiste à indiquer :

- à l'intérieur du cadre de la fonction :
  * le nom de la fonction
  * les paramètres formels avec leur type associé
  * éventuellement les entrées physiques, les sorties physiques et les variables globales (même si elles sont déconseillées)
- à l'extérieur du cadre de la fonction
  * les paramètres effectifs
  * les connexions des entrées physiques, des sorties physiques et des variables globales.

Exemple de représentation avec passage de paramètres

Exemple général



Prototype :

Il permet de définir l'ordre de passage des paramètres

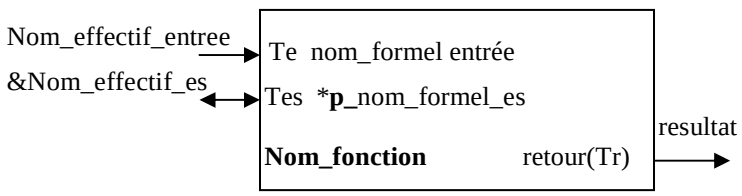
4- Utilisation d'une fonction

Utiliser une fonction consiste à :

- à placer le composant dans la séquence de traitement

- connecter le composant, c'est-à-dire appeler la fonction avec un passage de paramètres.
  La connexion des paramètres effectifs sur les paramètres formels se fait en respectant l'ordre des paramètres formels dans la fonction prototype et en remplaçant chaque paramètre formel par le paramètre effectif correspondant.
- exploiter éventuellement le résultat fourni par la fonction.

Exemple



type_retour Nom_fonction (type nom_formel_entrée, type *ptr_nom_formel_es)

resultat = Nom_fonction (nom_effectif_entrée, &nom_effectif_es)

5- Codage

Le codage consiste à traduire la description graphique en programme en langage C.
La démarche que nous proposons permet d'automatiser la traduction de la connexion graphique ou littérale formelle en instructions du langage C à partir de la description :

- graphique
- littérale formelle

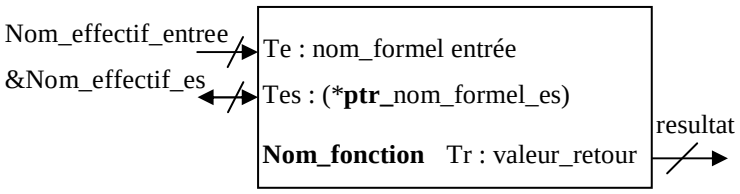
Graphique	
Littérale formelle	**Fonction prototype :** type_retour Nom_fonction (type nom_formel_entrée, type *ptr_nom_formel_es) **Connexion littérale formelle** Resultat = Nom_fonction (nom_effectif_entree, &nom_effectif_es)

Tableau de passage de paramètres par valeur

Le passage de paramètres par valeur se limite aux types simples parce que si le paramètre formel est de type composé (tableau ou structure) alors il est passé par adresse.

Type	Paramètre formel	Paramètre effectif
simple	Nom_formel_entree	Constante (valeur)
		Nom_effectif

Utilisation des paramètres passés par valeur dans la fonction

Les paramètres effectifs passés par valeur ne sont pas modifiés par la fonction appelée.
Cependant, le paramètre formel est considéré comme une variable locale dont le contenu peut être modifié localement dans la fonction. Autrement dit, cette modification n'est pas répercutée à l'extérieur de la fonction.
Le nom de la variable locale est celui du paramètre formel.

Tableau de passage de paramètres par adresse

Ce tableau récapitule les passages de paramètres par adresse. Il permet de faire des remplacements systématiques

type		**Paramètre formel**	**Paramètre effectif**
simple		*ptr_nom_formel	&nom_effectif
			Nom_pointeur_initialisé
composé	Tableau à une dimension	*ptr_nom_formel	Nom_tableau_effectif
			&Nom_tableau_effectif[0]
			Nom_pointeur initialisé sur tableau_effectif de même type
		Nom_formel_tableau[]	Nom_tableau_effectif
			Nom_pointeur initialisé sur tableau_effectif de même type
	Tableau à deux dimensions	Nom_tableau[][m_max]	Nom_tableau_effectif
			&Nom_tableau_effectif[0][0]
			Nom_pointeur initialisé

Utilisation des paramètres passés par adresse dans la fonction

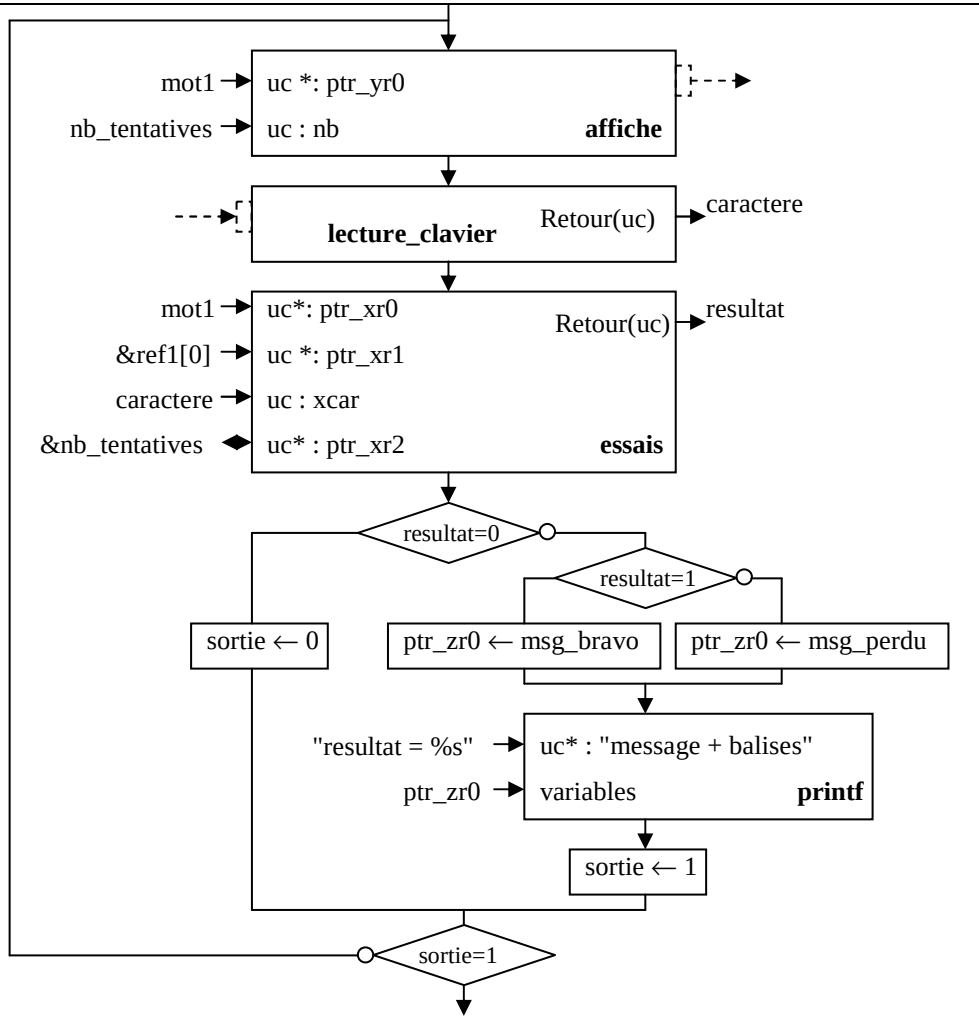
1- Ce tableau récapitule les passages de paramètres par adresse. Il montre comment les utiliser à l'intérieur de la fonction.

2- Volontairement, nous n'avons traité que les cas simples pour permettre à la majorité des étudiants de comprendre ces mécanismes.

3- Lorsqu'on manipule les pointeurs, il est conseillé d'utiliser les parenthèses pour préciser la portée de l'opérateur * car on n'a pas toujours à l'esprit la priorité entre les opérateurs.

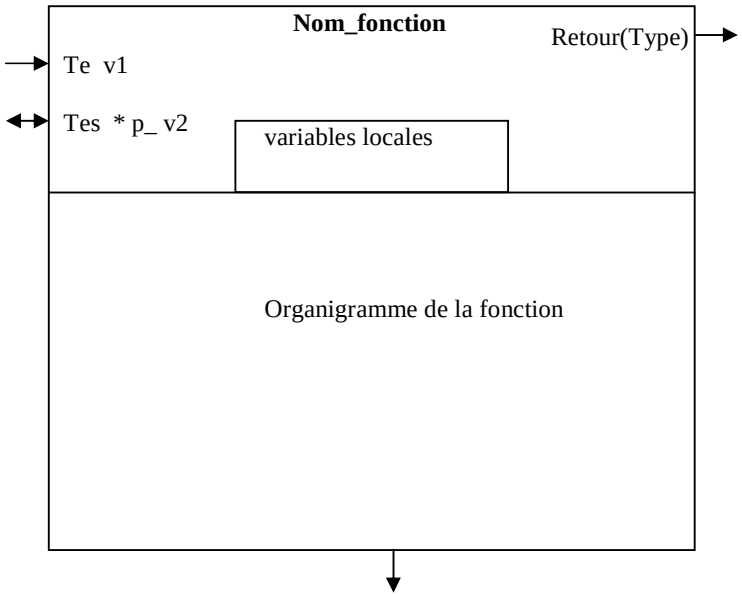
type		**Paramètre formel**	**Contenu de la variable pointée**	**Adresse de la variable pointée**
simple		*ptr_nom_formel	*ptr_nom_formel	ptr_nom_formel
composé	Tableau à une dimension	*ptr_nom_formel	*(ptr_nom_formel + index)	(ptr_nom_formel + index)
		Nom_formel_tableau[]	Nom_formel_tableau[index]	&Nom_formel_tableau[index]
	Tableau à deux dimensions	Nom_tableau[][m_max]	Nom_tableau[i][j]	&Nom_tableau[i][j]

Exemple de traduction d'un organigramme en langage C

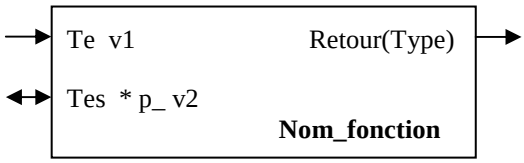



```
do
  {
    affiche(mot1, nb_tentatives);


    caractere = lecture_clavier();


    resultat = essais( mot1, &ref1[0], caractere, & nb_tentatives)

    if (resultat ==0)
        {   sortie  = 0;
        }
    else
        {
            if (resultat == 1)
                {   ptr_zr0 = msg_bravo;
                }
            else
                {   ptr_zr0 = msg_perdu;
                }

            printf("resultat= %s", ptr_zr0);

            sortie = 1;
        }
  }
 while (!(sortie==1));
```

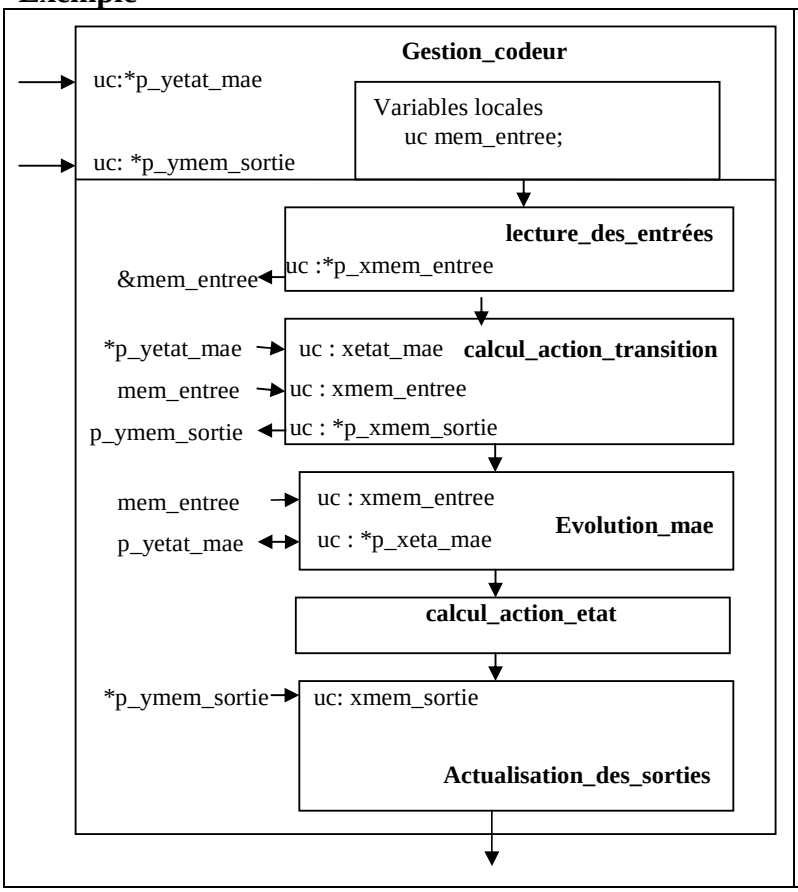
Présentation d'une fonction en organigramme



Vue extene de la fonction



Exemple




```
void gestion_couleur(
        unsigned char  *p_yetat_mae,
        unsigned char   *p_ymem_sortie)
{
  unsigned char mem_entree;

  lecture_des_entrees(&mem_entree);

  calcul_action_transition(
        *p_yetat_mae, mem_entree,
         p_ymem_sortie);

  Evolution_mae(mem_entree,
                        p_yetat_mae);

  calcul_action_etat();


  Actualisation_des_sorties(*p_tmem_sortie);

}
```

Langage C (partie 3)

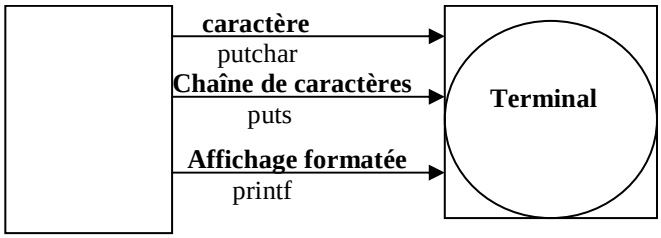
Fonctions d'entrées-sorties standard en langage C

Représentation sous forme de composants logiciels

Fonctions de sortie standard

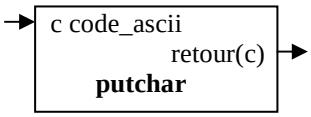
La sortie standard du langage C est un écran que nous appellerons terminal
On peut répartir les fonctions de sortie standard en 3 groupes de base :

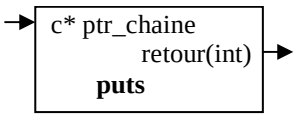
Affichage :
- d'un caractère
- d'une chaîne de caractères
- formaté : on peut choisir la façon d'afficher tout type d'information.



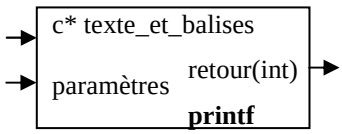
Notation de type pour les composants
c = char uc = unsigned char
i = int ui = unsigned int
f = float

Fonctions	Exemples

Affichage d'un caractère
 
 	
Prototype :
char putchar(char code_ascii)
	a) Afficher le caractère A sur l'écran
	putchar('A');
	
	b) Afficher le caractère donc le code ascii est dans la variable, car, de type char
	putchar(car);

Affichage d'une chaîne de caractères
 
 	
Prototype :
int puts(char* ptr_chaine)
	a) Afficher la chaîne de caractères "TEST" sur l'écran
	puts("TEST");
	
	b) Afficher la chaîne de caractères stockée dans la variable, tab_car, de type tableau de char
	puts(tab_car);
	
	Remarque importante:
	On doit s'assurer que la chaîne, tab_car, se termine par 0x00. Sinon la fonction affichera des caractères erronés.

Affichage formaté



Prototype :
int printf ("texte_et_balises", paramètres)

Remarques :

1- Texte est une chaîne constante de caractères.

2- "paramètres" est la liste des variables à visualiser séparées par des virgules.

3- Une balise est une indication de la façon d'interpréter l'information binaire à afficher. Il est noté % suivi d'un caractère d'interprétation

- %c = interprété comme un caractère
- %s = interprété comme une chaîne de caractères
- %u = interprété comme un entier non signé
- %d = interprété comme un entier signé
- %x = interprété comme un entier non signé et visualisé en hexadécimal
- %f = interprété comme un nombre en flottant visualisé sous la forme xxx.yyy

Pour les balises %u, %d, %x on peut préciser le nombre de digits à afficher.

a) les digits manquants sont remplacés par des espaces, si le nombre compte moins de digits à afficher,
syntaxe : %nu %nd %nX ou %nx

b) les digits manquants sont remplacés par des 0, si le nombre compte moins de digits à afficher.
syntaxe : %0nu %0nd %0nX ou %0nx

c) %f
syntaxe : %n.mf n digits pour la partie entière
m digits pour la partie fractionnaire

4- La balise doit être compatible avec le type d'information à afficher. On peut cependant utiliser la conversion explicite (Cast) pour forcer la compatibilité. Par exemple afficher un unsigned char avec la balise %u peut produire un résultat curieux en fonction du compilateur. Certains compilateurs assurent des conversions implicites correctes et d'autres pas. En effet, %u est associé aux entiers donc pour forcer la compatibilité, il suffit de faire un cast de la variable en écrivant (unsigned int) nom_variable.

5- Certains compilateurs ne supportent pas trop de paramètres dans une seule fonction printf. Il est conseillé de le faire en plusieurs printf.

6- Quelques caractères spéciaux utiles :

- \n aller à la ligne suivante
- \r aller au début de la ligne en cours
- \0x00 fin de la chaîne de caractères

Exemples

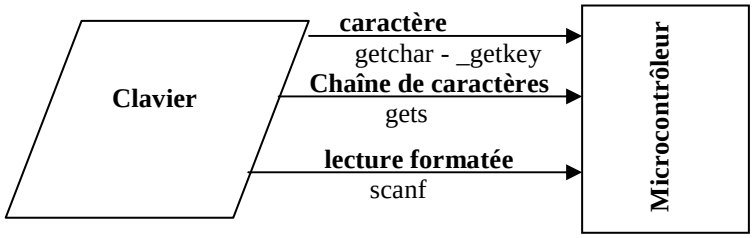
Exemples printf(suite)

Fonctions d'entrée standard

L'entrée standard du langage C est un clavier.
On peut répartir les fonctions d'entrée standard en 3 groupes .

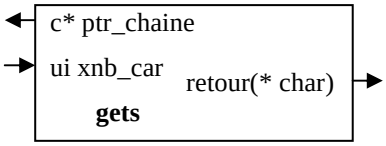
Lecture :

- d'un caractère
- d'une chaîne de caractères
- formatée :



Fonctions	Exemples
Lecture d'un caractère tapé au clavier sans écho (sans affichage sur l'écran du symbole du caractère tapé). **La fonction est bloquante.** C'est-à-dire qu'elle attend jusqu'à ce qu'un caractère soit tapé. retour(c) **_getkey** Prototype : char _getkey(void)	
Lecture d'un caractère tapé au clavier avec écho (avec affichage sur l'écran du symbole du caractère tapé) **La fonction est bloquante.** C'est-à-dire qu'elle attend jusqu'à ce qu'un caractère soit tapé. retour(c) **getchar** Prototype : char getchar(void)	
Indication de caractère disponible provenant du clavier **La fonction est non bloquante.** C'est-à-dire qu'elle n'attend pas qu'un caractère soit tapé. retour(bit) **kbhit** Prototype : bit kbhit(void)	

Lecture d'une chaîne de caractères tapés au clavier



Prototype :
char* gets(char* ptr_chaine,
unsigned int xnb_car)

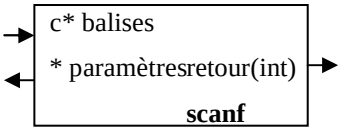
On doit indiquer :
- la chaîne de caractères où les codes ascii des caractères tapés doivent être stockés
- le nombre de caractères acceptés

La fin de saisie de la chaîne intervient si on tape :
Enter , Espace

Si on tape :
- moins de caractères que prévus, la fonction retourne une chaîne de caractères avec insertion automatique de \0x00 (fin de la chaîne)
- plus de caractères que prévus, la fonction tronque à xnb_car -1 et chaîne de caractères avec insertion automatique de \0x00 (fin de la chaîne)

Il est donc de votre responsabilité de prévoir suffisamment de places par rapport au nombre xnb_car.

Lecture des informations formatées



Prototype :
int scanf ("balises", paramètres)

Cette fonction fonctionne assez mal pour le formatage des informations. La moindre erreur de saisie crée des décalages de gestion du buffer d'entrée. Par ailleurs, comme la longueur de la chaîne n'est pas précisée, on risque des débordements mémoires.

Pour lire les informations formatées, nous conseillons d'utiliser la fonction gets et les fonctions de conversions comme :

atoi convertir une chaîne de caractères en un entier signé ou non

atof convertir une chaîne de caractères en un float