Polycopiés

Cours Informatique embarquée

GEII Toulouse 2020 - 2021

A. Nketsa

CHAP 5 Langage C (suite)

Langage C (suite 1)

- **Retour sur les pointeurs**

Langage C (suite 2)

- **Fonctions**
- **Notion de composant logiciel**
- **Organigramme structuré basé composants**
- **Exemples de passage de paramètres**
- **Exemple de traduction d'organigrammes en langage C**

Langage C (suite 3)

- **Fonctions d'entrées-sorties standard en langage C**
- **Représentation sous forme de composants logiciels**
- **Exemples d'utilisation des fonctions d'entrée-sortie standard**
  - ☞ **sorties standard**
  - ☞ **entrées standard**

Langage C (suite 4)
(Utilisation des masques en informatique embarquée)

- **Principe général**
- **Gestion des capteurs**
- **Actualisalisation des sorties**

CHAP 5 Langage C (Suite)

Langage C (suite 1)

- Retour sur les pointeurs

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

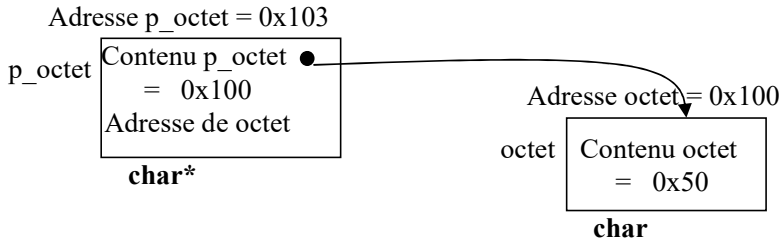
5) **Exemple :**

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

Schéma

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

5-1) **Schéma**



p_octet = &octet; ⇒ (p_octet) = 0x100

lecture : le contenu de p_octet = adresse de octet donc (p_octet) = 0x100

octet = (*p_octet) ⇒ (*p_octet) ≡ ((0x100)) = (octet) = 0x50

lecture : (*p_octet) est le contenu de la variable de type char donc l'adresse est dans p_octet

c'est-à-dire : le contenu de la variable ayant l'adresse 0x100

donc *p_octet = ((p_octet))

(adresse)

contenu

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**

5-5) **Aide pour manipuler les pointeurs**

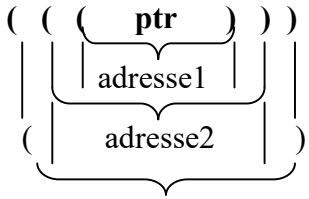
Pour manipuler les pointeurs, nous conseillons la notation parenthésée qui consiste à représenter le contenu par des parenthèses.

- (nom_variable) représente le contenu de nom_variable
- ((nom_pointeur)) représente le contenu du contenu de nom_pointeur

Pour simplifier la lecture et l'interprétation, la dernière paire de parenthèses ( ) représente **le contenu** de la variable pointée. Donc toutes les doubles parenthèses réprésentent **l'adresse**

Donc

((ptr)) ≡ *ptr	**((ptr)) ≡ (adresse)** adresse : (ptr) contenu variable pointée : (adresse)

(((ptr))) ≡ *(*ptr)	

d) Exemples d'application

unsigned char octet1 = 0x20;
unsigned char* p_uchar = &octet1;

Tableau de réservation des variables

adresse	Nom_variable	contenu	Remarque
0x50	octet1	0x20	Octet1 contient la valeur initiale 0x20
0x51			
0x52	p_uchar	0x50	p_uchar contient l'adresse de la variable pointée
0x53			

Déterminer les contenus des variables notées

variable	Contenu parenthésé	Contenu intermédiaire	Contenu manipulée
octet1	(octet1) = 0x20		octet1 = 0x20
p_char	(p_char) = 0x50		p_octet = 0x50
*p_char	*p_char = ((pchar))	(p_char) = 0x50 ((p_char)) = (0x50) = 0x20	*pchar = 0x20

e) Règles de manipulation 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 la 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];
unsigned int *p_uint;
	p_uint = tab_ui; // p_uint reçoit l'adresse de tab_ui
	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)

Les Fonctions en langage C

Une fonction est une entité qui peut être autonome.
C'est un élément important de la structuration d'un programme.
On peut considérer qu'une fonction est un composant comme en électronique numérique

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 souvent notée **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 globale

Nous interdisons de ne pas utiliser les variables globales dans les fonctions saf cas très particulier

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
}
```

Prototype d'une fonction :

```
void      nom_fonction (déclaration des paramètres formels);
```

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**

Prototype d'une fonction :

```
type_retour      nom_fonction (déclaration des paramètres formels)
```

g) Utilisation (appel) d'une fonction

g-1) L'utilisation de la fonction consiste à :

- à appeler la fonction en remplaçant les paramètres formels par les paramètres effectifs.

- à utiliser éventuellement le résultat de la fonction

Le paramètre effectif est la variable ou la constante qui remplace le paramètre formel.

g-2) Passage de paramètres

Le remplacement du paramètre formel par le paramètre effectif est appelé **passage de paramètres**

Ce passage doit respecter certaines règles :

- l'ordre des paramètres formels
- le type des paramètres formels et effectifs : ces types doivent être compatibles

g-3) Syntaxe d'utilisation

Deux cas :

Pas de retour, l'appel sera

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

Avec un retour, l'appel sera

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

Composant logiciel en langage C

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.

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

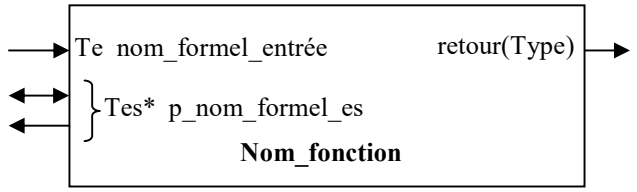
☞ **chaque paramètre formel possède une flèche pour indiquer le sens du paramètre**

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

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-

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

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

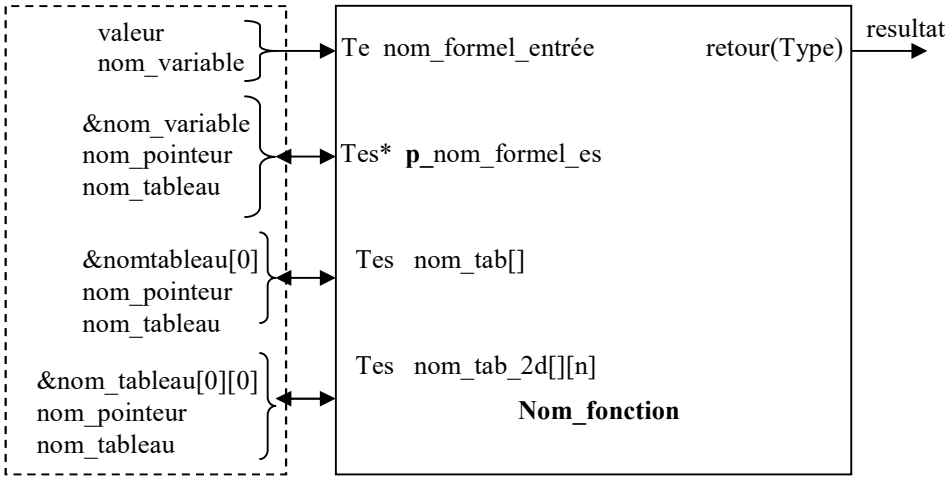
Schéma de principe du passage de paramètres

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

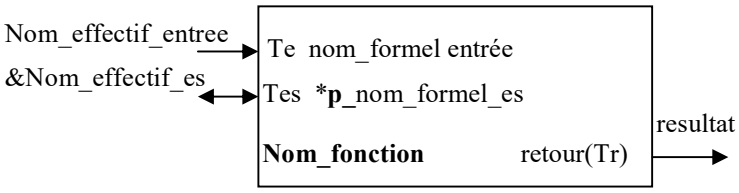
Utilisation d'une fonction

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.

Exemples



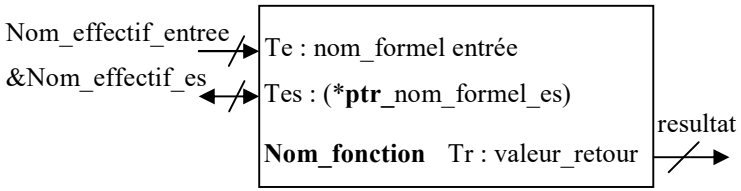
type_retour Nom_fonction (type nom_formel_entrée, type *ptr_nom_formel_es)

resultat = Nom_fonction (nom_effectif_entrée, &nom_effectif_es)

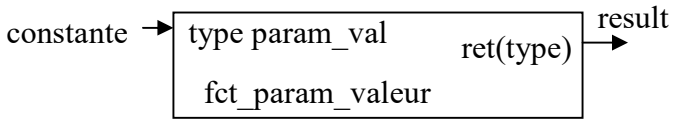
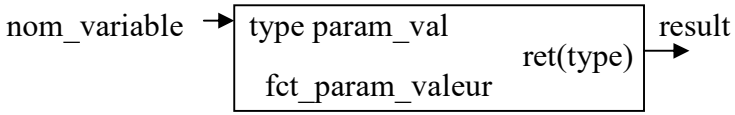
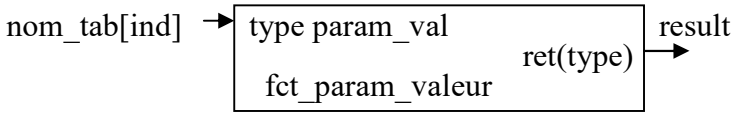
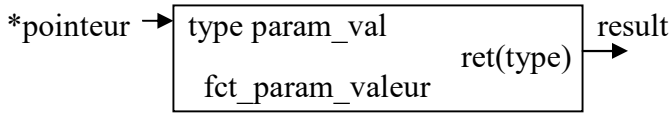
Codage

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 :

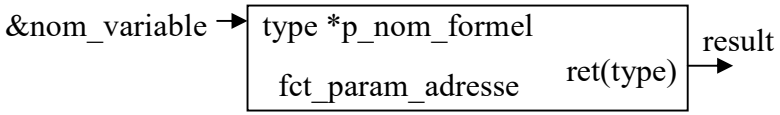
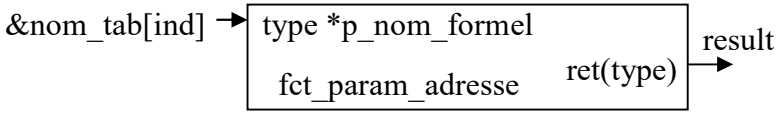
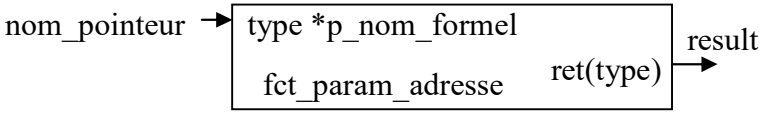
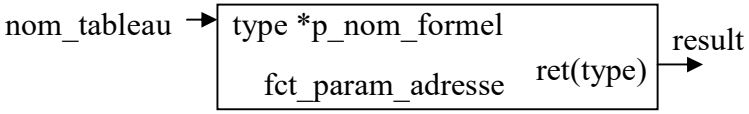
- graphique
- littérale formelle

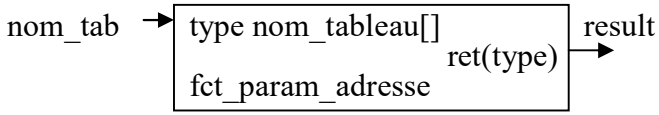
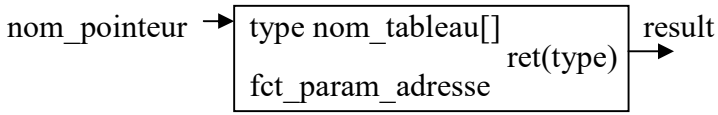
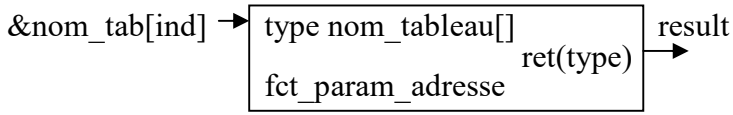
Graphique	
Littérale formelle	**Fonction prototype :** type_retour Nom_fonction (type nom_formel_entrée, type *ptr_nom_formel_es) **Connexion littérale formelle** Resultat = Nom_fonction (nom_effectif_entree, &nom_effectif_es)

Exemples de passage de paramètres par valeur

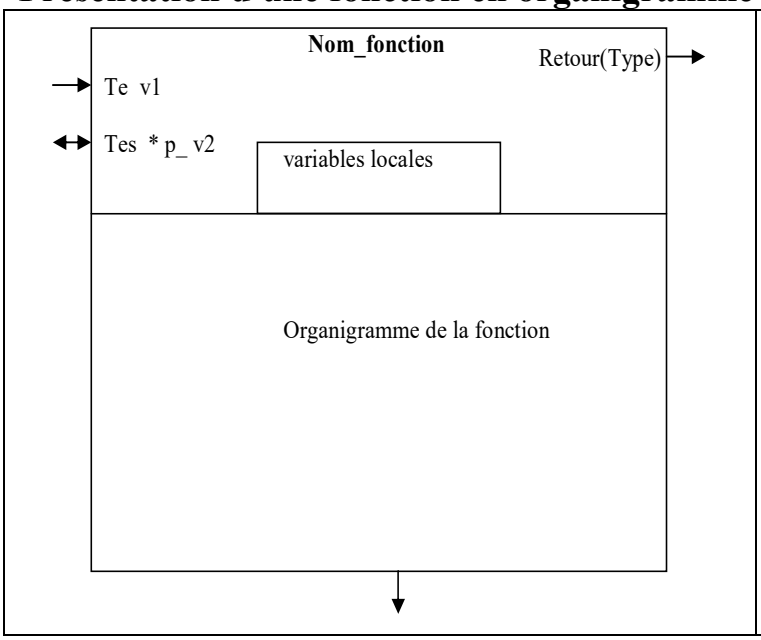
schéma	Codage en langage C
Le paramètre effectif est une constante  prototype type fct_param_valeur(type param_val);	result = fct_param_valeur(constante);
Le paramètre effectif est une variable  prototype type fct_param_valeur(type param_val);	result = fct_param_valeur(nom_variable);
Le paramètre effectif est une case d'un tableau  prototype type fct_param_valeur(type param_val);	result = fct_param_valeur(nom_tab[ind]);
Le paramètre effectif est une variable pointée  prototype type fct_param_valeur(type param_val);	result = fct_param_valeur(*pointeur);

Exemples de passage de paramètres par adresse **Il faut toujours passer une adresse**

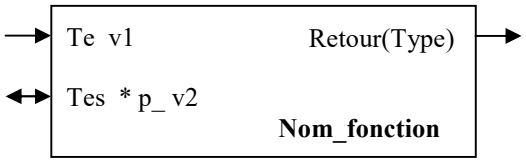
Format paramètre formel	Paramètre effectif	Schéma et codage en C
Type *p_nom_formel	Une variable Nom_variable	**Schéma**  **Prototype :** type fct_param_adresse(type *p_nom_formel); **Codage en C :** result = fct_param_adresse(&nom_variable);
	Une case d'un tableau nom_tab[ind]	**Schéma**  **Prototype :** type fct_param_adresse(type *p_nom_formel); **Codage en C :** result = fct_param_adresse(&nom_tab[ind]);
	Variable pointée *nom_pointeur	**Schéma**  **Prototype :** type fct_param_adresse(type *p_nom_formel); **Codage en C :** result = fct_param_adresse(nom_pointeur);
	Un tableau Nom_tableau	**Schéma**  **Prototype :** type fct_param_adresse(type *p_nom_formel); **Codage en C :** result = fct_param_adresse(nom_tableau);

Type nom_tableau[]	Un tableau Nom_tab	**Schéma**  **Prototype :** type fct_param_adresse(type nom_tableau[]); **Codage en C :** result = fct_param_adresse(nom_tab);
	Variable pointée *nom_pointeur	**Schéma**  **Prototype :** type fct_param_adresse(type nom_tableau[]); **Codage en C :** result = fct_param_adresse(nom_pointeur);
	Une case d'un tableau nom_tab[ind]	**Schéma**  **Prototype :** type fct_param_adresse(type nom_tableau[]); **Codage en C :** result = fct_param_adresse(&nom_tab[ind]);

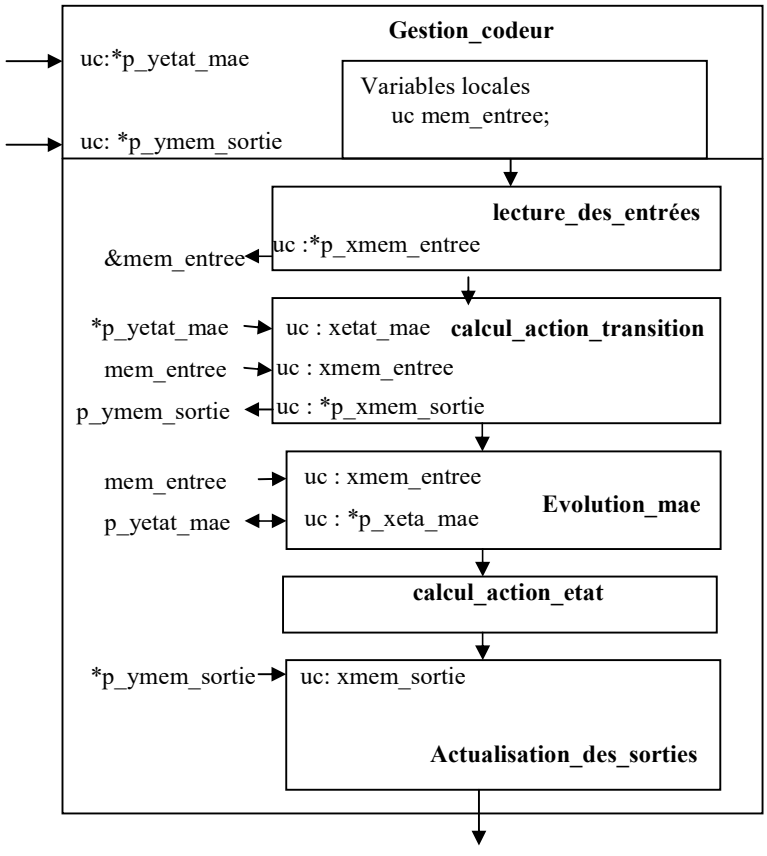
Présentation d'une fonction en organigramme



Vue extene de la fonction



Exemple




```
void gestion_codeur(
        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
  );

}
```

Les entrées sorties standard

Remarques importantes

a) Les instructions d'entrées sorties standard

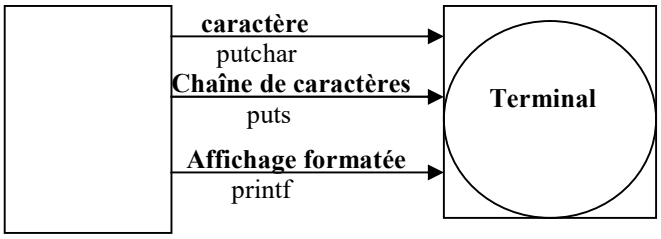
	En C++ **C'est le compilateur qui fait presque tout pour vous pour afficher**	**En C standard** **C'est vous qui devez dire au compilateur ce que vous souhaitez afficher**
Sortie standard	**Cout**	**Ecriture d'un caractère** putchar(char code_ascii) **Ecriture d'une chaîne de caractères** puts(char *p_ch)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

Fonctions d'entrées-sorties standard en langage C

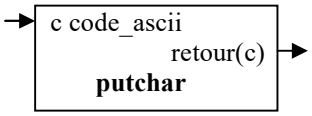
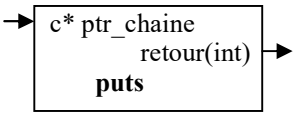
Fonctions de sortie standard

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 :

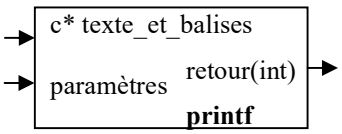
Affichage :
- d'un caractère
- d'une chaîne de caractères
- formaté : on peut choisir la façon d'afficher tout type d'information.



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 Déclaration char tab_car[20]; **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

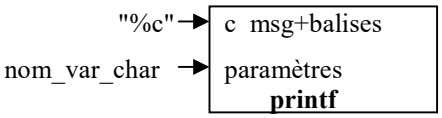
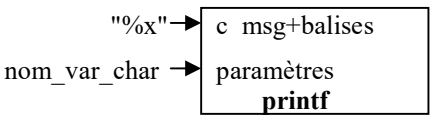
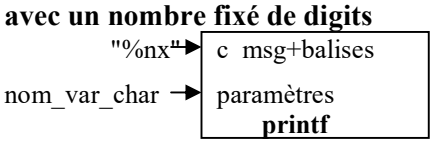
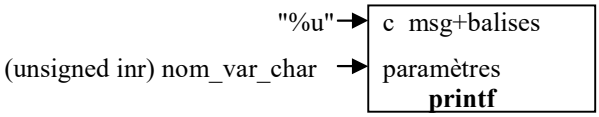
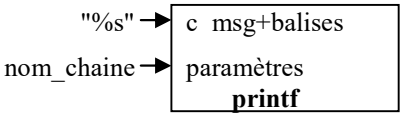
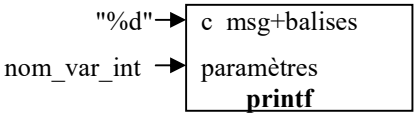
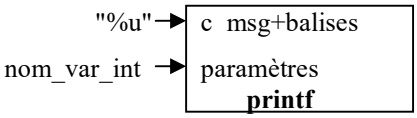
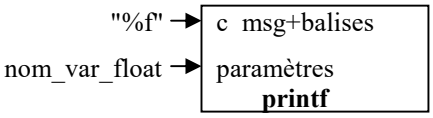
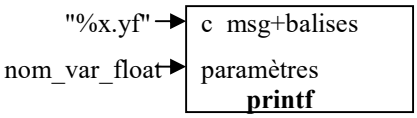
4- La balise doit être compatible avec le type d'information à afficher. On peut cependant utiliser la conversion explicite (Cast) pour forcer la compatibilité.
Par exemple afficher un unsigned char avec la balise %u peut produire un résultat curieux en fonction du compilateur. Certains compilateurs 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 positionner le curseur à la ligne suivante en consevant la colonne
- \r positionner le curseur au début de la ligne en cours, donc colonne 0
- \0x00 fin de la chaîne de caractères
- \g bip sonore

Exemples printf

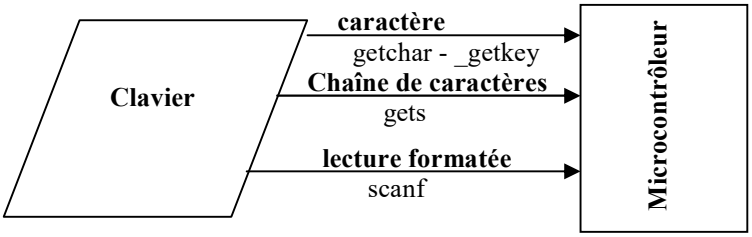
Affichage d'une variable de type char en char 	**printf("%c", nom_var_char);**
Affichage d'une variable de type char en hexadécimal 	**printf("%x", nom_var_char);**
Affichage d'une variable de type char en hexadécimal avec un nombre fixé de digits 	**printf("%nx", nom_var_char);**
Affichage d'une variable de type char en entier non signé 	**printf("%u", (unsigned int) nom_var_char);**
Affichage d'une variable de type chaine de caractères 	**printf("%s", nom_chaine);**
Affichage d'une variable de type int en entier signé 	**printf("%d", nom_var_int);**
Affichage d'une variable de type int en entier non signé 	**printf("%u", nom_var_int);**
Affichage d'une variable de type float en float 	**printf("%f", nom_var_float);**
Affichage d'une variable de type float en float avec un nombre fixé de digits 	**printf("%x.yf", nom_var_float);**

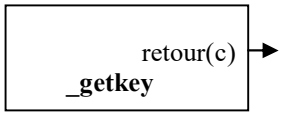
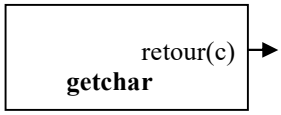
Fonctions d'entrée standard

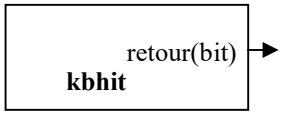
L'entrée standard du langage C est un clavier.
On peut répartir les fonctions d'entrée standard en 3 groupes .

Lecture :

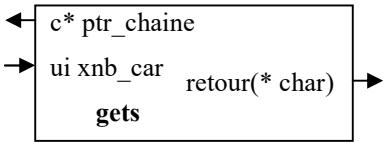
- d'un caractère
- d'une chaîne de caractères
- formatée :



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.  Prototype : char _getkey(void)	var_char = _getkey();
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.  Prototype : char getchar(void)	var_char = getchar();

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.
 
Prototype :
bit kbhit(void)
	if (kbhit())
	{ //lire la touche détectée
	_getkey(); ou getchar();
	}

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

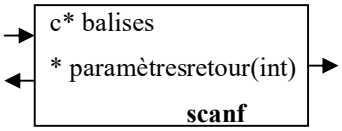
La fin de saisie de la chaîne intervient si on tape :
Enter , Espace

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.

gets(nom_chaine, nb_car);

Lecture des informations formatées



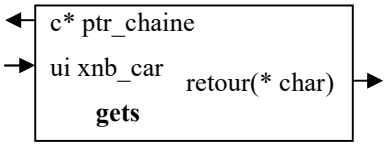
Prototype :
int scanf ("balises", paramètres)

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.

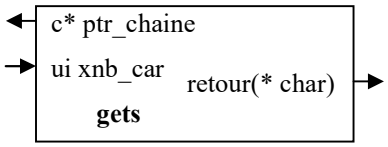
Pour lire les informations formatées, nous conseillons d'utiliser la fonction gets et les fonctions de conversions comme :

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

atof convertir une chaîne de caractères en un float

Lecture d'un entier tapé au clavier
 
 	
Prototype :
char* gets(char* ptr_chaine,
unsigned int xnb_car)
	gets(nom_chaine, 5);
	entier = atoi(nom_chaine);
	
	Remarque :
	S'il y a une erreur, par exemple la saisie d'une lettre, dans la chaîne, atoi retourne 0 sans indiquer d'erreur.

Lecture d'un entier tapé au clavier avec vérification des chiffres	

Lecture d'un float tapé au clavier
 
 	
Prototype :
char* gets(char* ptr_chaine,
unsigned int xnb_car)
	gets(nom_chaine, 8);
	entier = atof(nom_chaine);
	
	Remarque :
	S'il y a une erreur, par exemple la saisie d'une lettre, dans la chaîne, atoi retourne 0 sans indiquer d'erreur.

Lecture d'un float tapé au clavier	

Langage C (suite 4)
(Utilisation des masques en informatique embarquée)
- Principe général
- Gestion des capteurs
- Actualisalisation des sorties

En informatique embarquée, les capteurs et les acyionneurs binaires sont souvent regoupés pour formés des mots (octet ou mot de 16bits)
Lors de l'utilisation, on a souvent besoin de ne tester ou manipuler qu'un ensemble de ces bits.
Pour cela : il faut les isoler pour les manipuler
Nous avons vu lors des instructions logiques que l'on peut isoler un ou plusieurs bits dans un mot, de même en utilisation les instructions booléennes on peut tester ces bits

Rappel
Pour isoler les bits ,
- on construit un masque en ET
- et on effectue l'opération logique ET (&)

Pour tester les bits isolés,
- on construit un masque avec la valeur attendue des bits à tester
- on effectue une comparaison entre le résultat des bits isolés et le masque des valeurs attendues

Pour mettre des bits à 0
- on construit un masque en ET des bits à mettre à 0
- on effectue un ET logique (&) entre le mot et le complément à 1 (~) du masque en ET

Pour mettre des bits à 1
- on construit un masque en ET des bits à mettre à 1
- on effectue un OU logique (|) entre le mot et le masque en ET

Pour compémenter des bits à 1
- on construit un masque en ET des bits à complémenter
- on effectue un OU logique (|) entre le mot et le masque en ET

Gestion des capteurs

Considérons que nous avons un octet qui regroupe : les capteurs, les interruopteurs et les poussoirs d'un système occupant dans l'octet les positions suivantes :

Variable : octet_entree

poussoirs		interrupteurs		capteurs			
B7							B0
Pouss1	Pouss0	Inter1	Inter0	C3	C2	C1	C0

On admet que les capteurs sont actifs à 0

Gestion des actionneurs

Considérons que nous avons un octet qui regroupe : les actionneurs d'un système occupant dans l'octet les positions suivantes :

Variable : octet_sortie

poussoirs		interrupteurs		capteurs			
B7							B0
buzzer	sirène	moteur1	moteur0	Led3	Led2	Led1	Led0

On admet que les actionneurs sont actifs à 1

Exercice

si c2=0 et inter0=1 et pouss0 = 0
alors allumer la Led2
sinon eteindre la led2 et complémenter la led3

Donc il faut isoler les bits C2, inter0 et poussà dans octet_entree

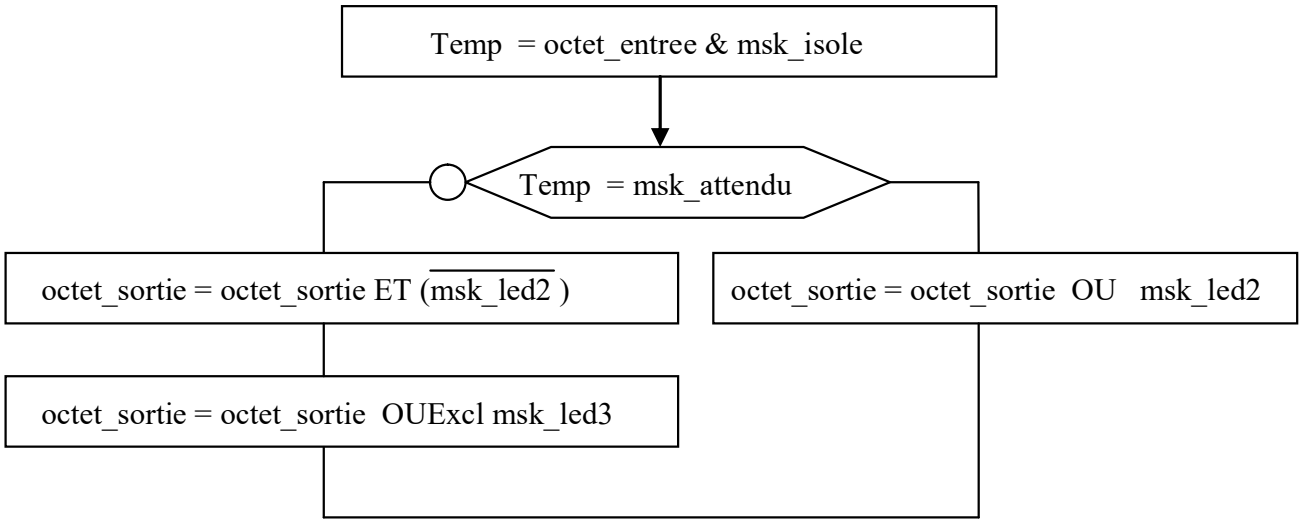
	b7							b0
	Pouss1	Pouss0	Inter1	Inter0	C3	C2	C1	C0
Masque en ET (isolement) Msk_isole	0	1	0	1	0	1	0	0
Masque valeur attendue Msk_attendu	0	0	0	1	0	0	0	0

- masque en ET : 01010100b ≡ 0x54
- masque de test : 00010000b ≡ 0x10

Pour allumer la led2 sans modifier les autres, il faut construire le masque correspondant

	b7							b0
	buzzer	sirène	moteur1	moteur0	Led3	Led2	Led1	Led0
Masque en ET pour mise à 1 led2 (msk_led2)	0	0	0	0	0	1	0	0
Masque pour complémenter led3 (msk_led3)	0	0	0	0	1	0	0	0

D'où le programme



Traduction en langage C

```
Temp = octet_entree & msk_isole;
if (Temp == msk_attendu)
  {
     octet_sortie = octet_sortie | msk_led2;
  }
Else
  {
      octet_sortie = octet_sortie & (~msk_led2);
      octet_sortie = octet_sortie ^ msk_led3;
   }
```