Langage C

Par K. SYLLA

EPT-2010

Khalifasylla@gmail.com

Historique

- Défini une première fois par Kernighan et Ritchie des laboratoires Bell dans le livre « The C progamming language » paru en 1978
- Puis, avec le succès du système UNIX écrit en C, plusieurs compilateurs sont sur le marché
- Normalisé à la fin des années 80 par l'ANSI*, plus précisément en 1989
- En 1990, l'ISO** adopta la norme ANSI comme norme ISO

*American National Standards Institute

**International Standards Organization

Présentation

Le Langage C est un langage de bas niveau dans le sens où il a accès à des données telles que des bits, des adresses…

Il permet donc l'écriture de systèmes d'exploitation et de logiciels.

C dispose également de structures de base nécessaires à la conception d'applications structurées, ce qui en fait aussi un langage de haut niveau

Il est l'un des premiers langages offrant des possibilités de programmation modulaire

Plan

- Syntaxe
- Types et variables
- Opérateurs, expressions
- Instructions de contrôle
- Fonctions et procédures
- Tableaux et pointeurs
- Structures, unions et énumérations
- Entrées/Sorties
- Compilation

I - Généralités sur la syntaxe

Le format du texte est libre et la mise en page n'a aucune signification pour le compilateur. Elle permet une bonne lisibilité du programme. Néanmoins, certaines règles constituent la syntaxe de base du langage.

Donnons la structure générale d'un programme:

programme:

liste de déclarations de variables

liste de déclaration de fonctions ou de procédures

Les déclarations des variables peuvent être omises mais il est nécessaire de donner au moins la définition d'une fonction particulière *main()*

Généralités sur la syntaxe

Structure d'un programme

```
#inclusion de sources
…
int i,j;                    // Déclarations de variables
float a,b;                  //      ..
void p(int k)               //      Déclaration de procédures
{…
}
int f(float x, float y)     //  Déclaration de fonctions.
{…
}
int main()                  //  Fonction particulière
{…
}
```

Nous allons voir les commentaires et les inclusions de sources

Généralités sur la syntaxe

Les commentaires en C

Ils détaillent le programme et contribuent à la lisibilité et à la compréhension de celui-ci.

Les lignes de code placées entre ses signes ne sont pas pris en compte par le compilateur.

Ils débutent par /* et se termine par */ sur plusieurs lignes.

Exemple /*Ceci est un commentaire*/

Les commentaires ne peuvent être imbriqués.

Généralités sur la syntaxe

Inclusion de sources

Il est fréquent d'avoir besoin d'une partie déjà implémentée dans une autre unité de compilation, ou même d'avoir besoin de fonctions de la bibliothèque standard. L'inclusion de source nous permet alors d' y accéder.

La commande est la suivante

#include <nom du fichier> ou *#include « nom du fichier »*

Cette commande comme nous l'avons vue, est placée en début du programme.

Cette ligne sera remplacée par le pré-processeur qui mettra à cette place le fichier correspondant.

Exemple:

```
#include <stdio.h>
```

Généralités sur la syntaxe

Les unités lexicales

Maintenant que nous avons la structure générale d'un programme, nous allons voir les types d'unités lexicales qu'utilisent le langage.

Il s'agit principalement

- des identificateurs
- des mots clés
- de constantes
- de chaînes
- d'opérateurs
- et de signes de ponctuation

Nous ne verrons dans cette section que les deux premiers types.

Généralités sur la syntaxe

Les identificateurs

Ils permettent de nommer les entités du programme (variables, fonctions,…)

Règles de construction d'un identifiant:

- il doit être pris à partir de l'alphabet :'a - z' 'A - Z' '0 - 9' '_'
- doit commencer par une lettre ou par _.
- la longueur ne doit pas dépasser 31 pour les identificateurs internes (dans ce cas, le compilateur tronque les caractères significatifs)
- La distinction entre majuscules et minuscules n'est pas garantie pour les identificateurs externes

i, Premier, _d, h1, varLocale sont des identificateurs valides.

Contrairement à 1i, i!h, p:1

Généralités sur la syntaxe

Les mots réservés

Ils sont propres au langage et ne peuvent être pris comme identificateurs.

- Définitions des données
  char const double float int long short signed unsigned void volatile
- Classes d'allocation
  auto extern static register
- Constructeurs
  enum struct typedef union
- *I*nstructions de boucles
  do for while
- Sélections
  case default else if switch
- Ruptures de séquences
  break continue goto return
- Divers
  sizeof

II – Types et variables

C est un language ***fortement typé*** : Chaque donnée en C (constantes, variables) a un type.

Ce type est trés important et donne la place en mémoire et le format de la représentation.

II – Types et variables

Chaque donnée en C (constantes, variables) a un type.

Ce type est trés important et donne la place en mémoire et le format de la représentation.

Les types de bases

Nous allons d'abord voir les types de données élémentaires, nous reviendront sur les types complexes (Tableaux, structures, types personnalisés, pointeurs…)

II – TYPES

Le type *void*:

Type vide. (fonctions sans arguments ou sans valeurs de retour, pointeurs)

Les types entiers

Le type *char* ou *unsigned char* (1 octet):

Il permet de représenter un caractère. C'est un nombre entier qui est le code ASCII* du caractère. Il peut aussi être défini comme *unsigned*.

Nous disposons du type wchar_t dans le fichier <stddef.h> qui gère des alphabets de plus de 255 caractères

*ASCII = American Standard Code for Information Interchange

Khalifasylla@gmail.com

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	|	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

II – TYPES(entiers)

Le type *short* ou *signed short* (2 octets):

Stocke des entiers : -32768 à 32767

Unsigned short: 0 à 65535

II – TYPES(entiers)

Le type *short* ou *signed short* (2 octets):

Stocke des entiers : -32768 à 32767

Unsigned short: 0 à 65535

Le type *long* ou *signed long* (4 octets):

Entiers de -2.147.843.648 à -2.147.843.647

Unsigned long: 0 à 4.294.967.295

Khalifasylla@gmail.com

II – TYPES(entiers)

Le type *short* ou *signed short* (2 octets):

Stocke des entiers : -32768 à 32767

Unsigned short: 0 à 65535

Le type *long* ou *signed long* (4 octets):

Entiers de -2.147.843.648 à -2.147.843.647

Unsigned long: 0 à 4.294.967.295

Le type *int* ou *signed int :*

C’est le type entier et peut correspondre au long ou au short. Généralement il se confond au short et est donc représenté sur 2 octets.

II – TYPES(réels)

Le type *float* (4 octets):

23 bits pour la mantisse, 8 bits pour l'exposant, 1 bit pour le signe.

Permet de représenter des réels : 3.4 *10 exp(-38) à 3.4*10 exp(38)

II – TYPES(réels)

Le type *float* (4 octets):

23 bits pour la mantisse, 8 bits pour l'exposant, 1 bit pour le signe.

Permet de représenter des réels : 3.4 *10 exp(-38) à 3.4*10 exp(38)

Le type *double* (8 octets):

52 bits pour la mantisse, 11 bits pour l'exposant, 1 bit pour le signe.

Permet de représenter des réels : 1.7 *10 exp(-308) à 1.7*10 exp(308)

Le type récent long double représenté sur 10 octets.

II – TYPES(constantes)

Constantes de type entier.

Elles peuvent être entrées sous forme décimale, octale ou hexadécimale.

- Les constantes décimales s'écrivent de la manière usuelle (ex 455)
- Les constantes octales commencent par un 0 (ex 0577)
- Les constantes hexadécimales par 0x ou 0X (ex 0x9a2f)

II – TYPES(constantes)

Constantes de type entier.

Elles peuvent être entrées sous forme décimale, octale ou hexadécimale.

- Les constantes décimales s'écrivent de la manière usuelle (ex 455)
- Les constantes octales commencent par un 0 (ex 0577)
- Les constantes hexadécimales par 0x ou 0X (ex 0x9a2f)

455u, 0577U, 0x9a2fu pour dire qu'ils sont sans signe (unsigned)

455L, 0577l, 0x9a2fL pour spécifier qu'ils sont de type long

II – TYPES(constantes)

Constantes de type entier.

Elles peuvent être entrées sous forme décimale, octale ou hexadécimale.

- Les constantes décimales s'écrivent de la manière usuelle (ex 455)
- Les constantes octales commencent par un 0 (ex 0577)
- Les constantes hexadécimales par 0x ou 0X (ex 0x9a2f)

455u, 0577U, 0x9a2fu pour dire qu'ils sont sans signe (unsigned)

455L, 0577l, 0x9a2fL pour spécifier qu'ils sont de type long

Constantes de type réel.

Mantisse et Exposant.	En C	En mathématique
	2 . correspond	à 2
	.3	à 0.3
	5.4	à 5.4
	2e7 ou 2.e7	à 2×10^{7}
	2.4e-5	à 2.3×10^{-4}

Khalifasylla@gmail.com

II – TYPES(constantes)

Constantes de type caractère.

Une constante caractère s'écrit entre apostrophe ' (exemple 'j').

Attention à 1 et '1'.

Cependant certains caractères peuvent avoir des fonctionnalités en C, alors nous ajouterons le signe \ pour signifier cette fonctionnalité.

Pour écrire le caractère \ donc nous mettrons '\\'.

Caractères non imprimables

II – TYPES(constantes)

Illustrer chaque caractère avec un petit programme.

Séquence	Signification
\n	nouvelle ligne
\t	tabulation horizontale
\v	tabulation verticale
\b	retour d’un caractère en arrière
\r	retour chariot
\f	saut de page
\a	signal sonore
\ddd	affiche un caractère de code ASCII ddd en octal
\xddd	affiche un caractère de code ASCII ddd en hexadécimal

II – TYPES(constantes)

Constantes de type chaînes de caractère.

Une constante caractère s’écrit entre guillemets ‘‘Ceci est une chaîne’’.

Différence entre ‘d’ et ‘‘ d’’ ?

II – TYPES(constantes)

Constantes de type chaînes de caractère.

Une constante caractère s'écrit entre guillemets ‘‘Ceci est une chaîne’’.

Différence entre ‘d’ et ‘‘ d’’ ?

#define identificateur valeur_de_la_constante

II –Variables

Une variable est une donnée du programme qui peut être modifiée.

Une variable est définie par un nom, et par un type.

Elle est chargée en mémoire.

II –Variables

Une variable est une donnée du programme qui peut être modifiée.

Une variable est définie par un nom, et par un type.

Elle est chargée en mémoire.

Définition:

Il faut nécessairement définir une variable avant de l'utiliser. Lors de la définition, on lui attribue un nom et un type.

TYPE nomvar_1, nomvar_2;

Exemple:

int i, j;

Float a,b,c;

II –Variables

Une variable est une donnée du programme qui peut être modifiée.

Une variable est définie par un nom, et par un type.

Elle est chargée en mémoire.

Définition:

Il faut nécessairement définir une variable avant de l'utiliser. Lors de la définition, on lui attribue un nom et un type.

TYPE nomvar_1, nomvar_2;

Exemple:

int i, j;

Float a,b,c;

Le nom d'une variable est un identificateur donc doit respecter les règles s'y appliquant.

II –Variables

Opérations sur les variables.

Initialisation:

Comme son nom l'indique, il s'agit ici de donner dés la définition une valeur à la variable.

int i=1;

II –Variables

Opérations sur les variables.

Initialisation:

Comme son nom l'indique, il s'agit ici de donner dés la définition une valeur à la variable.

int i=1;

Affectation:

Semblable à l'initialisation, elle permet d'assigner une valeur à une variable à n'importe quelle étape du programme.

II –Variables

Opérations sur les variables.

Initialisation:

Comme son nom l'indique, il s'agit ici de donner dés la définition une valeur à la variable.

int i=1;

Affectation:

Semblable à l'initialisation, elle permet d'assigner une valeur à une variable à n'importe quelle étape du programme.

Lors de l'affectation et de l'initialisation, il faut attribuer à la variable une valeur de même type.

Impression formatée

Khalifasylla@gmail.com

Impression formatée

Sortie formatée:

Contrôler la forme et le format des données à afficher sur la sortie standard.

Utiliser la fonction **printf().**

Syntaxe printf(’’format’’, argument_1, argument_2, -----, argument_3);

Impression formatée

Sortie formatée:

Contrôler la forme et le format des données à afficher sur la sortie standard.

Utiliser la fonction **printf().**

Syntaxe printf(’’format’’, argument_1, argument_2, -----, argument_3);

La chaîne qui donne le format est composée de % et d’une lettre qui donne le type de l’argument à afficher.

On peut vouloir afficher plusieurs arguments en même temps, alors spécifier le format autant de fois.

Impression formatée

Sortie formatée:

Contrôler la forme et le format des données à afficher sur la sortie standard.

Utiliser la fonction **printf().**

Syntaxe printf(’’format’’, argument_1, argument_2, -----, argument_3);

La chaîne qui donne le format est composée de % et d’une lettre qui donne le type de l’argument à afficher.

On peut vouloir afficher plusieurs arguments en même temps, alors spécifier le format autant de fois.

Affichage d’entiers décimaux:

Pour afficher un nombre **décimal entier** il faut utiliser la lettre d.

```
int i=20;
printf(’’%d’’, i);
```

Impression formatée

Sortie formatée:

Contrôler la forme et le format des données à afficher sur la sortie standard.

Utiliser la fonction **printf().**

Syntaxe printf(’’format’’, argument_1, argument_2, -----, argument_3);

La chaîne qui donne le format est composée de % et d’une lettre qui donne le type de l’argument à afficher.

On peut vouloir afficher plusieurs arguments en même temps, alors spécifier le format autant de fois.

Affichage d’entiers décimaux:

Pour afficher un nombre **décimal entier** il faut utiliser la lettre d.

```
int i=20;
printf(’’%d’’, i);      printf(’’%d’’, 4+6);
```

Impression formatée

Sortie formatée:

Contrôler la forme et le format des données à afficher sur la sortie standard.

Utiliser la fonction **printf().**

Syntaxe printf(’’format’’, argument_1, argument_2, -----, argument_3);

La chaîne qui donne le format est composée de % et d’une lettre qui donne le type de l’argument à afficher.

On peut vouloir afficher plusieurs arguments en même temps, alors spécifier le format autant de fois.

Affichage d’entiers décimaux:

Pour afficher un nombre **décimal entier** il faut utiliser la lettre d.

```
int i=20;                                   int i,j;  i=1 ;j=4;
printf(’’%d’’, i);      printf(’’%d’’, 4+6);        printf(’’i – j =%d’’, i – j );    printf(’’%d – %d =%d’’,  i, j, i – j ) ;
```

Impression formatée

Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les types donnés comme non signés

Affichage d'entiers octaux:

Prendre la lettre o.

Affichage d'entiers hexadécimaux:

Prendre la lettre x ou X.

printf(''Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X'', 10,10,10);

Impression formatée

Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les types donnés comme non signés

Affichage d'entiers octaux:

Prendre la lettre o.

Affichage d'entiers hexadécimaux:

Prendre la lettre x ou X.

printf(''Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X'', 10,10,10);

Pour vérifier que les types doivent être non signés prendre un nombre négatif

Exemple -1 et 65535.

Impression formatée

Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les types donnés comme non signés

Affichage d'entiers octaux:

Prendre la lettre o.

Affichage d'entiers hexadécimaux:

Prendre la lettre x ou X.

printf(''Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X'', 10,10,10);

Pour vérifier que les types doivent être non signés prendre un nombre négatif

Exemple -1 et 65535.

Affichage de caractères

Utiliser la lettre c.

```
char a='a';
printf('' %c '',a);
```

Impression formatée

Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les types donnés comme non signés

Affichage d'entiers octaux:

Prendre la lettre o.

Affichage d'entiers hexadécimaux:

Prendre la lettre x ou X.

printf(''Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X'', 10,10,10);

Pour vérifier que les types doivent être non signés prendre un nombre négatif

Exemple -1 et 65535.

Affichage de caractères

Utiliser la lettre c.

```
char a='a';
printf('' %c '',a);        printf(''Bonjour %c tout le monde '', 'à');
```

Impression formatée

Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les types donnés comme non signés

Affichage d'entiers octaux:

Prendre la lettre o.

Affichage d'entiers hexadécimaux:

Prendre la lettre x ou X.

printf(''Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X'', 10,10,10);

Pour vérifier que les types doivent être non signés prendre un nombre négatif

Exemple -1 et 65535.

Affichage de caractères

Utiliser la lettre c.

```
char a='a';                                                    char a='a';
printf('' %c '',a);        printf(''Bonjour %c tout le monde '', 'à');     printf(''le suivant de a est %c'', a+1);
```

Impression formatée

Afficher les 128 premiers caractères de la table Ascii, mis en forme.

Donner le code Ascii en décimal, octal, hexadécimal des 26 caractères majuscules de l’alphabet français.

Khalifasylla@gmail.com

Impression formatée

Afficher les 128 premiers caractères de la table Ascii, mis en forme.

Donner le code Ascii en décimal, octal, hexadécimal des 26 caractères majuscules de l'alphabet français.

Affichage de réels:

Les lettres f, e et g pour les types float et double et %Lf pour le type long double.

La lettre f est la plus utilisée.

```
float r=1.2;
printf(”%f”, r);
```

La fonction affichera 6 chiffres après la virgule

Pour les types qu'on vient de voir , il est possible d'ajouter un nombre entre % et la lettre pour donner la place que va prendre l'affichage.

On peut également intercaler h pour le type short, l pour long

Impression formatée

Afficher les 128 premiers caractères de la table Ascii, mis en forme.

Donner le code Ascii en décimal, octal, hexadécimal des 26 caractères majuscules de l'alphabet français.

Affichage de réels:

Les lettres f, e et g pour les types float et double et %Lf pour le type long double.

La lettre f est la plus utilisée.

```
float r=1.2;
printf(''%f'', r);
```

La fonction affichera 6 chiffres après la virgule

Pour les types qu'on vient de voir , il est possible d'ajouter un nombre entre % et la lettre pour donner la place que va prendre l'affichage.

On peut également intercaler h pour le type short, l pour long

Affichage de chaînes de caractères

Prendre la lettre s.

```
printf(''%s'', ''Bonjour à tout le monde'').
```

Tableau de formats pour printf

Symbole	Type décrit
%d	Entier (entier décimal)
%o	Entier (entier octal)
%x ou %X	Entier (entier hexadécimal)
%u	Entier (entier non signé)
%f	Réel
%e ou %E	Réel (format exponentiel)
%g ou %G	Réel (soit e ou f)
%c	Caractère
%s	Chaîne de caractère

Khalifasylla@gmail.com

Saisie formatée

Comme pour l’impression, il s’agit de contrôler le type de données que l’on veut saisir.

scanf().

Syntaxe scanf(’’format’’, argument_1, argument_2, -----, argument_3);

Même syntaxe mais différence fondamentale.

Saisie formatée

Comme pour l’impression, il s’agit de contrôler le type de données que l’on veut saisir.

scanf().

Syntaxe scanf(’’format’’, argument_1, argument_2, -----, argument_3);

Même syntaxe mais différence fondamentale.

Donc argument_1 représente l’adresse de l’argument correspondant.

scanf(’’%d ’’,&i)

Pour enregistrer, de l’entrée standard (clavier), la valeur de i.

%d	Entier décimal (int)	%u	Décimal (non signé)
%hd	Décimal (short)	%hu	Décimal (short non signé)
%ld	Décimal (long)	%lu	Décimal (long non signé)
%o	Entier octal (int)	%f	Réel (float)
%ho	Octal (short)	%lf ou %Lf	Réel (double, ou long d.)
%lo	Octal (long)	%e ou %E	Réel (float en exponentiel)
%x	Entier hexadécimal (int)	%le ou %LE	Réel (double ou long d.)
%hx	Hexadécimal (short)	%g ou %G	Réel(float)
%lx	Hexadécimal (long)	%lg ou %LG	Réel (double ou long d.)
%c	caractère	%s	Chaîne de caractères

Khalifasylla@gmail.com

III - Opérateurs et expressions

Nous allons commencer par étudier les opérateurs (et opérandes) car les expressions se définissent à partir d'eux.

III – 1- Opérateurs

Trois types d’opérateurs

- → Opérateurs unaires
- → Opérateur binaires
- → Opérateur ternaire

III – 1- Opérateurs

Trois types d'opérateurs

- Opérateurs unaires
  - Opérateur moins unaire : - inverse le signe
  - Opérateur plus unaire : + ne fait rien.
  - Opérateur d'incrémentation : ++ ex1 : i++ ←→ i=i+1 ; ex2 : j = ++i et j=i++;
  - Opérateur de décrémentation : --
  - Opérateur de taille : sizeof(.) ex : int i; sizeof(i)= ?
  - NON logique: ! ex : si EXP est vraie alors !EXP est fausse et inversement .
  - Complément à un : ~ s'applique au bits
  - Opérateur d'adresse : &
  - Opérateur d'indirection d'adresse : *
  - Opérateur de conversion ou cast : (type)EXP
- Opérateurs binaires
- Opérateurs ternaires

III – 1- Opérateurs

Trois types d'opérateurs

- → Opérateurs unaires
- → Opérateurs binaires
  - → Opérateurs arithmétiques:
    - → addition, soustraction : + , - ;
    - → multiplication, division : *, / ;
    - → reste de la division : %;
  - → Opérateurs de comparaison:
    - → inférieur, inférieur ou égal : <, <= ;
    - → supérieur, supérieur ou égal : >, >= ;
    - → égal, différent : = =, != ;
  - → Opérateur Logiques:
    - → Niveau bit: et, ou, ou exclusif : &, | , ^ ;
    - → Niveau expression : et, ou : &&, || ;
  - → Opérateur de décalage:
    - → Décalage vers la droite: >>
    - → Décalage vers la gauche: <<
  - → Opérateurs de succession: , ; exemple i=(j=7, k=2);
  - → Opérateur d'affectation : = ; peut être précédé de +,-,*,/,&,| , ^,<<,>>.
- → Opérateurs ternaires

III – 1- Opérateurs

Trois types d’opérateurs

- → Opérateurs unaires
- → Opérateurs binaires
- → Opérateur ternaire : Expression1 ? Exp2:Exp3. exemple : a>b?a:b

PRECEDENCE DES OPÉRATEURS

Classe d'opérateur	Opérateur(s)	Associativité
Parenthésage	()	de gauche à droite
Suffixes	() [] -> .	de gauche à droite
	++ --	
Un-aires	& * + - ~ !	de droite à gauche
	sizeof sizeof()	
Changement de type	(type)	de droite à gauche
Multiplicatifs	* / %	de gauche à droite
Additifs	+ -	de gauche à droite
Décalages	<< >>	de gauche à droite
Comparaisons	< <= > >=	de gauche à droite
Égalités	== !=	de gauche à droite
et bit à bit	&	de gauche à droite
ou exclusif bit à bit	^	de gauche à droite
ou bit à bit	|	de gauche à droite
et logique	&&	de gauche à droite
ou logique	||	de gauche à droite
Condition	?:	de droite à gauche
Affectations	= += -= *= /= %=	de droite à gauche
Affectations	&= |= ^=	
Affectations	<<= >>=	
Succession	,	de gauche à droite

Khalifasylla@gmail.com

III – 2 Expressions

Une Expression est une suite syntaxiquement correcte d’opérateurs et d’opérandes

Une expression renvoie toujours une valeur.

III – 2 Expressions

Une Expression est une suite syntaxiquement correcte d'opérateurs et d'opérandes

Une expression renvoie toujours une valeur.

Une instruction est une expression suivie de « ; » ou une instruction de contrôle.

Une instruction peut aussi être une suite d'instructions délimitées par les signes «{» et «}».

IV – Instructions de contrôle

→ Instructions répétitives

→ Instructions alternatives

→ Instructions de branchements

Khalifasylla@gmail.com

IV – Instructions de contrôle

→ Instructions répétitives

while, for, do while.

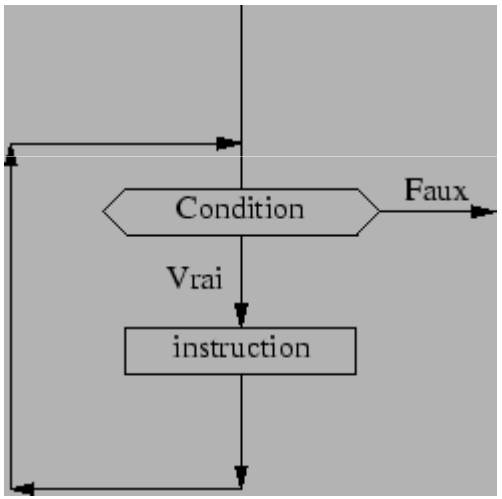
IV – Instructions de contrôle

→ Instructions répétitives

→ while

syntaxe : while (expression) instruction

déroulement →



IV – Instructions de contrôle

→ Instructions répétitives

→ for

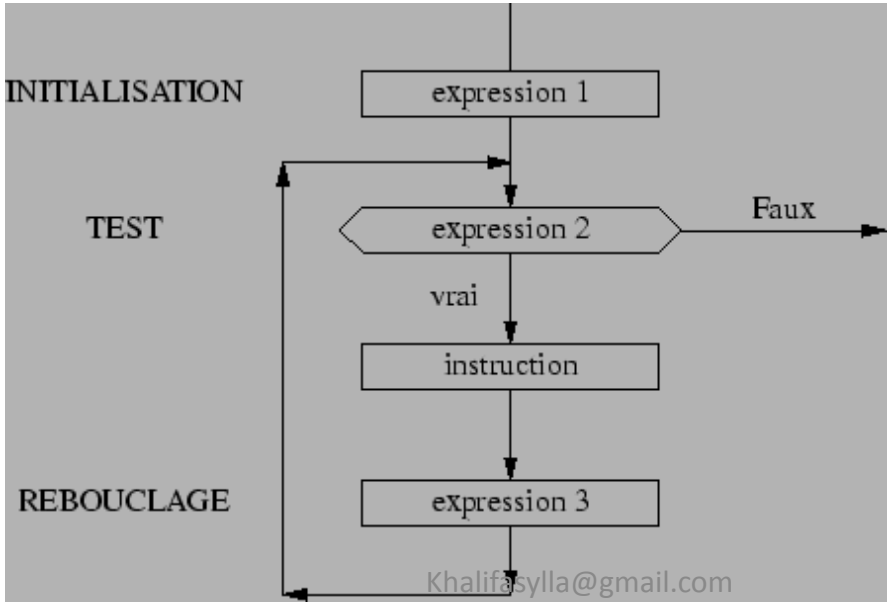
syntaxe : for(exp1;exp2;exp3) instruction

exp1 : instruction d'initialisation, effectuée une seule fois à l'entrée de la boucle

exp2 : test d'arrêt de la boucle, examiné avant chaque entrée de la boucle

exp3 : instruction d'incrémentation, effectuée en fin de boucle

Déroulement →



IV – Instructions de contrôle

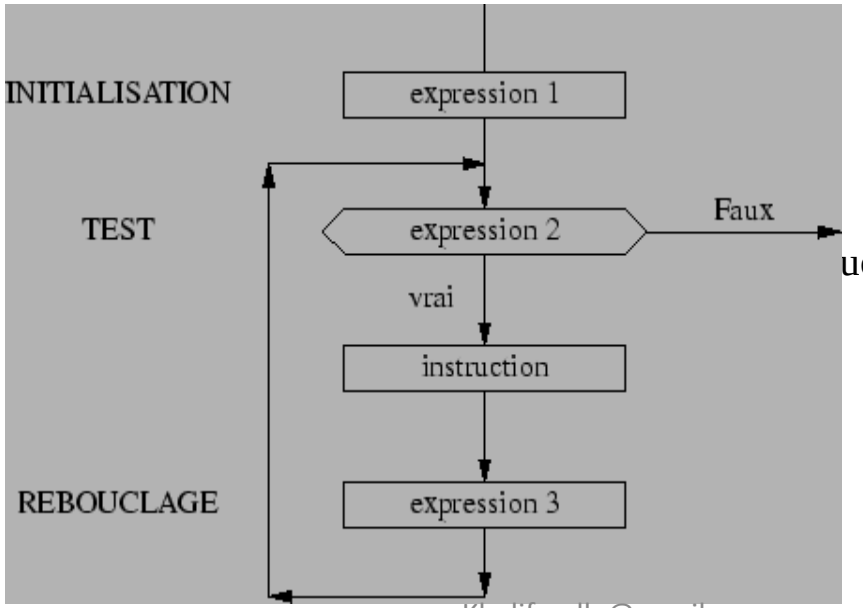
→ Instructions répétitives

→ for

syntaxe : for(exp1;exp2;exp3) instruction

exp1 : instruction d'initialisation, effectuée une seule fois à l'entrée de la boucle
exp2 : test d'arrêt de la boucle, examiné avant chaque entrée de la boucle
exp3 : instruction d'incrémentation, effectuée en fin de boucle

Déroulement →



for(e1;e2;e3) instruction

←→

e1;
while(e2)
{
uction
e3;
}

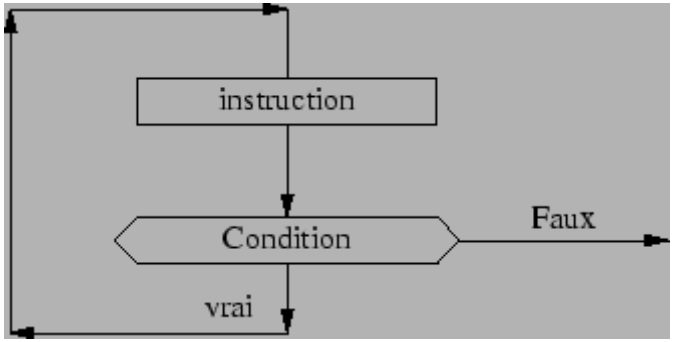
IV – Instructions de contrôle

→ Instructions répétitives

→ do while

syntaxe : do instruction while (expression);

Déroulement →



IV – Instructions de contrôle

→ Instructions alternatives

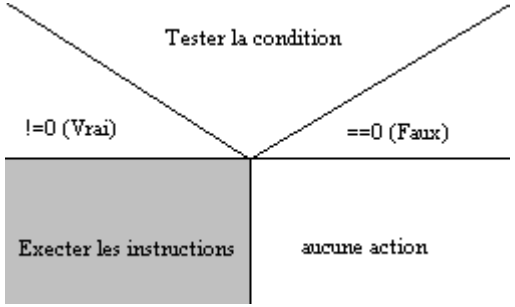
if – else ; switch – case – default

IV – Instructions de contrôle

→ Instructions alternatives

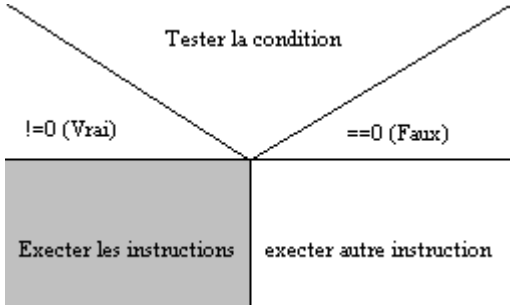
if – else

syntaxes : if(expression) instruction →



if(expression) instruction1 →

else instruction2



!!! On rappelle qu'une instruction peut être simple ou composée

IV – Instructions de contrôle

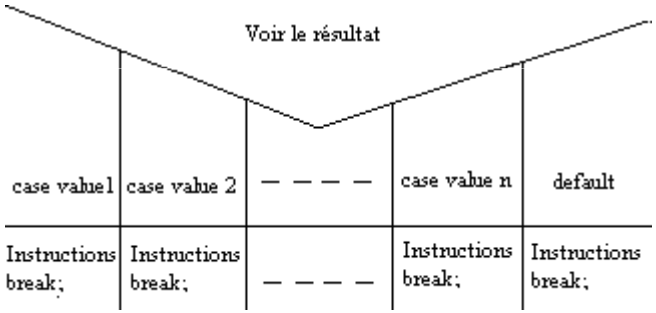
→ Instructions alternatives

switch – case – default

Syntaxe

```
switch(expression)
{
  case value1: inst1 inst2… instn break;
  case value2: ---------------------- break;
  -----
  -----
  case valueN:  ----------------------------
  default: ----------------------------------
 }
```

→



IV – Instructions de contrôle

→ Instructions de branchements

break, continue, goto, return

Khalifasylla@gmail.com

IV – Instructions de contrôle

→ Instructions de branchements

break et continue.

break permet de sortir complètement de la boucle dans laquelle il est utilisé.

continue fait sauter les instructions restantes de l'itération courante et passe à l'itération suivante.

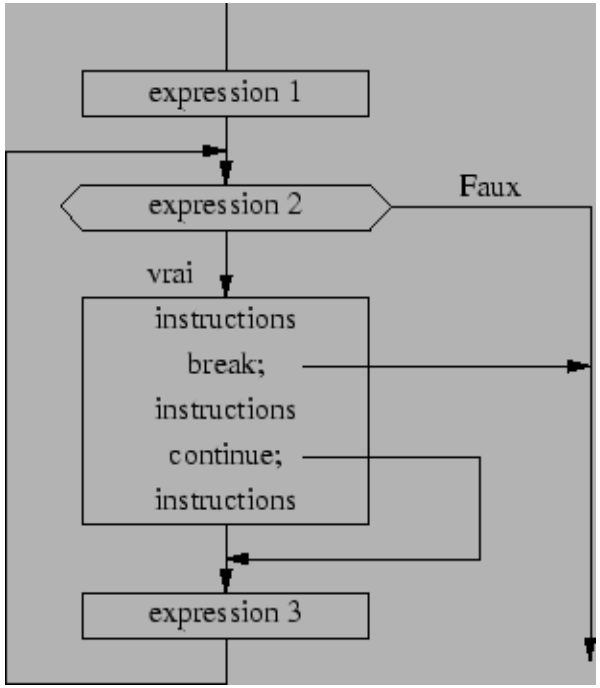
IV – Instructions de contrôle

→ Instructions de branchements

Break et continue.

break permet de sortir complètement de la boucle dans laquelle il est utilisé.

continue fait sauter les instructions restantes de l'itération courante et passe à l'itération suivante.



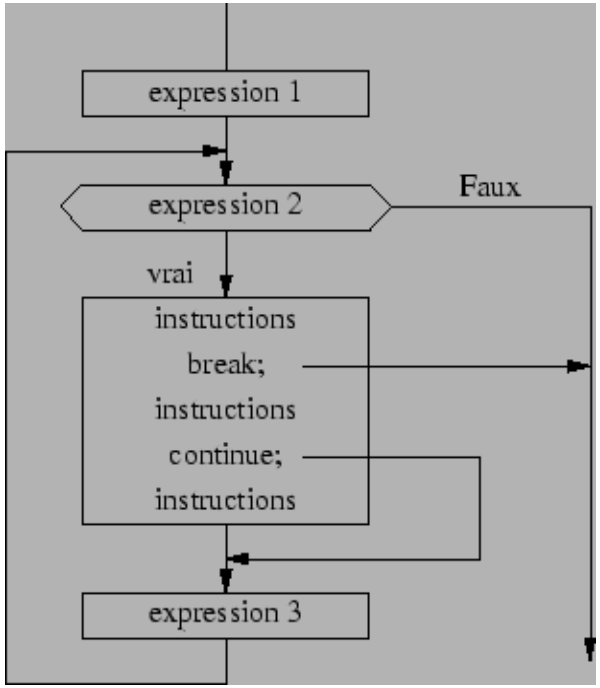
IV – Instructions de contrôle

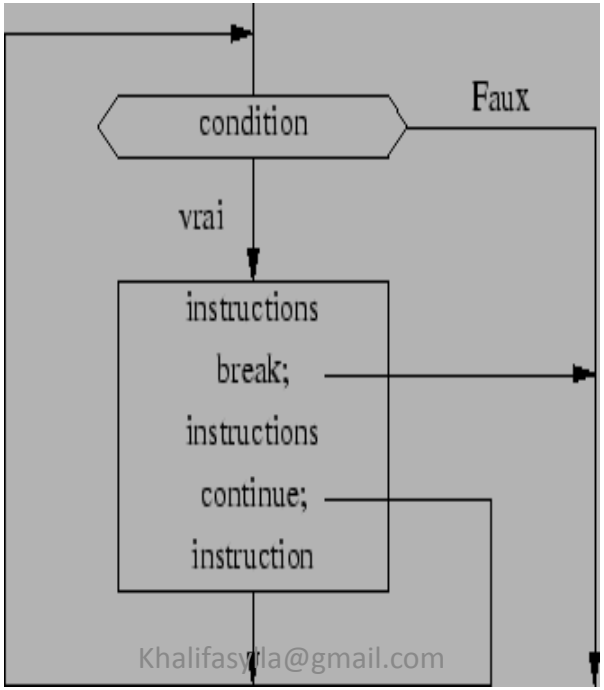
→ Instructions de branchements

Break et continue.

break permet de sortir complètement de la boucle dans laquelle il est utilisé.

continue fait sauter les instructions restantes de l'itération courante et passe à l'itération suivante.





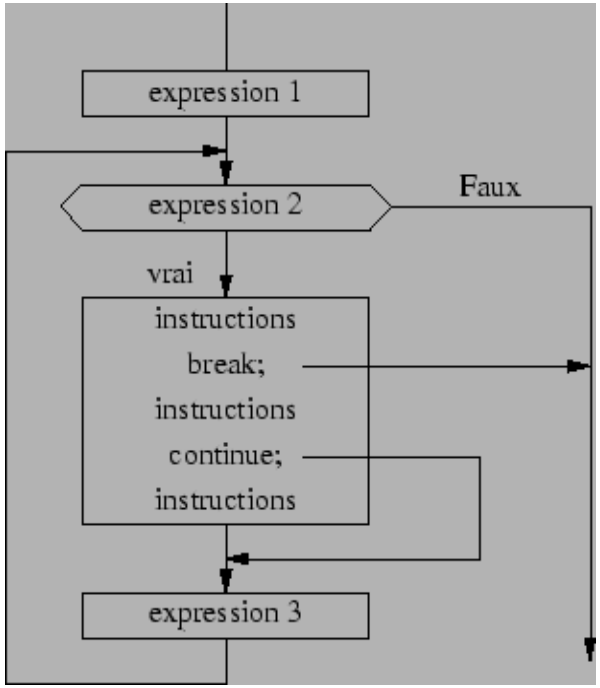
IV – Instructions de contrôle

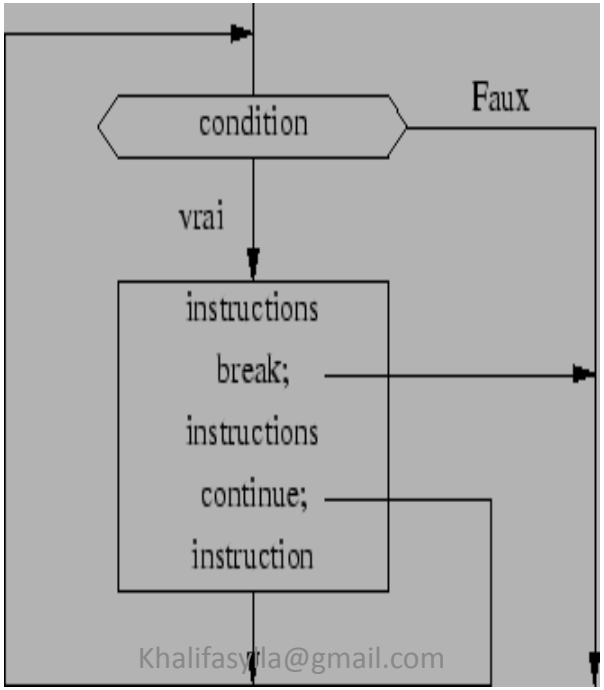
→ Instructions de branchements

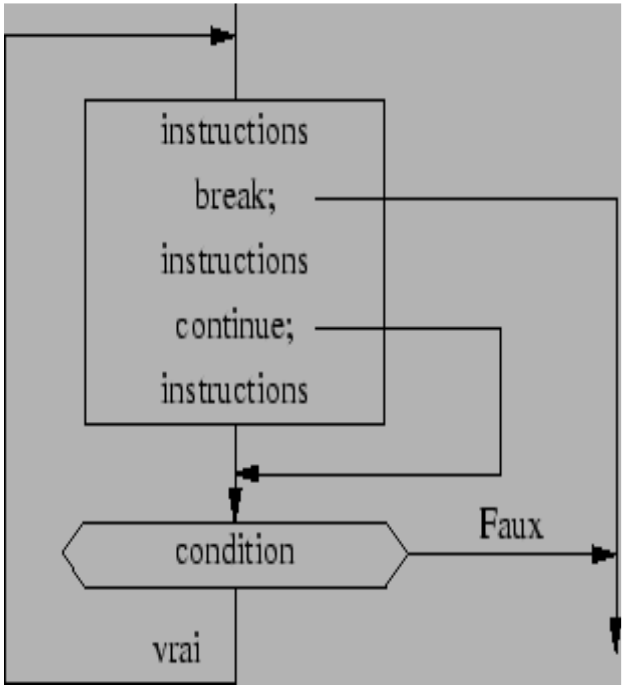
Break et continue.

break permet de sortir complètement de la boucle dans laquelle il est utilisé.

continue fait sauter les instructions restantes de l'itération courante et passe à l'itération suivante.







IV – Instructions de contrôle

→ Instructions de branchements

goto

syntaxe : goto label1;

---

label1:

provoque un saut à un endroit repéré par une étiquette (ou label).

Une étiquette est une chaîne suivie de « : ».