Programmation en langage C

EL Moukhtar ZEMMOURI

ENSAM – Meknès

Version 1.0 – 2015 / 2016

What is C

"C is a general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators. C is not a "very high level" language, nor a "big" one, and is not specialized to any particular area of application."

Kernighan & Richie, 1978

Sommaire

- Généralités
- Éléments de base de C
  - Identificateurs
  - Mots clés
  - Commentaires
  - Les types
- Lecture / écriture
- Structures de contrôle

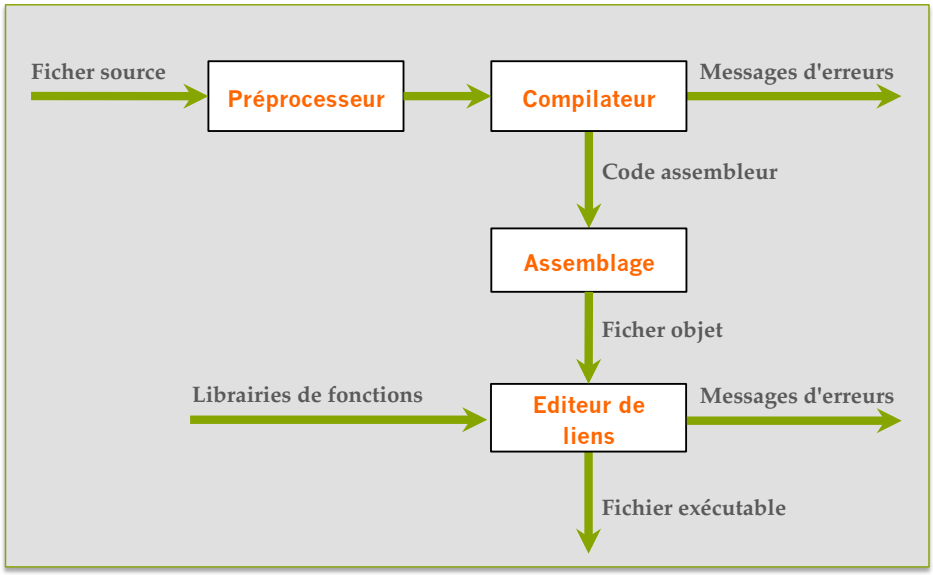
Un peu d'histoire …

- Le C a été conçu en 1972 par Dennis Richie et Ken Thompson
  - Deux chercheurs au Bell Labs
  - Objectif : développer un système d'exploitation UNIX
- En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C
  - Première édition du livre "The C programming language"
- En 1983, l'ANSI décida de normaliser le langage
  - 1989 la définition de la norme ANSI C (ou C89)
  - Deuxième édition de "The C programming language"
- L'ISO à repris la même norme en 1990 (ou C90)

La compilation

- C est un langage compilé
  - Par opposition au langages interprétés (Python, Matlab, ...)
- ➔ Un programme C écrit dans un fichier source est traduit en totalité en langage machine avant exécution.
- Quatre phases :
  - Traitement par le préprocesseur
  - Compilation
  - Assemblage
  - Édition de liens
- Exemples de compilateurs
  - GCC du projet GNU
  - MinGW (Minimalist GNU for Windows)

La compilation



Éléments de base du langage

Premier programme

- À éditer à l'aide d'un éditeur de texte
  - Fichier source hello.c
- Compiler
  - gcc hello.c
- Puis exécuter ☺

```c
#include <stdio.h>

main(){

    printf("Hello world !");
}
```

Composants élémentaires du C

- Six catégories de composants élémentaires :
  - Les identificateurs
  - Les mots clés
  - Les opérateurs
  - Les constantes
  - Les chaines de caractères
  - Les signes de ponctuation
- Les commentaires sont enlevés par le préprocesseur

Identificateurs

- Servent pour identifier (donner des noms) aux entités du programme :
  - variables, fonctions, types, ...
- Suite de caractères parmi :
  - les lettres (minuscules ou majuscules, non accentuées),
  - les chiffres,
  - le “blanc souligné _
- Commence par une lettre (ou _)
- Ne doit pas être un des mots clés du langage.

Mots clés

- 32 mots clefs de la l'ANSI C
- int float double char void short long signed unsigned struct enum union
- const volatile static auto register extern typedef
- if else for while do switch case default break continue goto
- return sizeof

Les commentaires

- Les commentaires permettent de documenter un programme.
  - Améliorer la lisibilité du programme
- Un commentaire en C commence par **/*** et se termine par ***/**
  - Exemple /* ceci est un commentaire */
- **//** permet un commentaire sur une seule ligne
- N.B. Les commentaires ne peuvent pas être imbriqués

Les types de base

- Le C est un langage typé :
  - ➔ Toute variable, constante ou fonction est d'un type précis
- Le type définit la représentation mémoire d'un objet
- Les types de base en C concernent
  - les entiers
  - les flottants (nombres réels)
  - les caractères

Les types entiers

- Le type entier représente l'ensemble des entiers relatifs (positifs et négatifs)
- Plusieurs sous types :

Type	Taille	Valeurs	Intervalle
char	8 bits	caractères	[–128 , 127]
short	16 bits	Entiers courts	[–32768 , 32767]
int	32 bits	Entiers	$[-2^{31} , 2^{31} - 1]$
long	64 bits	Entiers long	$[-2^{63} , 2^{63} - 1]$
unsigned char	8 bits	caractères	[0 , 255]
unsigned short	16 bits	Entiers courts non signés	[0 , 65536]
unsigned int	32 bits	Entiers non signés	$[0 , 2^{32} - 1]$
unsigned long	64 bits	Entiers long non signés	$[0 , 2^{64} - 1]$

N.B. La taille mémoire des types en C dépend de la machine !

Les types entiers

- **Exercice :**
  - Ecrire un programme qui donne d'intervalle des nombres représentés par les types : char, short, int, et long, signed et unsigned.

Les types flottants

- 3 types correspondant à différentes précisions :

Type	Taille	Valeurs
float	32 bits	Flottants simple précision
double	64 bits	Flottants double précision
long double	128 bits	Flottants précision étendue

- Erreur de troncature :
  - Quelle que soit la machine utilisée, on est assuré que cette erreur (relative) ne dépassera pas 10^{-6} pour le type float et 10^{-10} pour le type long double.

N.B. La taille mémoire des types en C dépend de la machine !

Les types flottants

- **Exercice :**
  - Ecrire un programme qui donne le plus grand nombre et le plus petit nombre proche de zéro (epsilon) qu'on peut représenter avec le type float.

Le type des caractères

- Le type **char** représente le jeu de caractères de la machine
- Un char en C est codé sur un octet (8 bits)
- Un caractère est encodé en utilisant un entier avant sa représentation binaire en mémoire.
- Plusieurs encodages sont utilisés:
  - ASCII (American Standard Code for Information Interchange)
    - Vesrion originale (1960) représente 128 caractères avec les nombres de 0 à 127 sur 7 bits.
    - Version étendue (1980) représente 256 sur 8bits
  - Unicode (1990) représente 65 536 sur 16 bits

Le type des caractères

- L'ensemble des caractère ASCII (Version originale)
  - Comment lire : R sur ligne 8 colonne 2 est encodé par 82.

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Le type des caractères

- Les caractères non imprimables :

NOTATION EN C	CODE ASCII (hexadécimal)	ABRÉVIATION USUELLE	SIGNIFICATION
\a	07	BEL	cloche ou bip (alert ou audible bell)
\b	08	BS	Retour arrière (Backspace)
\f	0C	FF	Saut de page (Form Feed)
\n	0A	LF	Saut de ligne (Line Feed)
\r	0D	CR	Retour chariot (Carriage Return)
\t	09	HT	Tabulation horizontale (Horizontal Tab)
\v	0B	VT	Tabulation verticale (Vertical Tab)
\\	5C	\	
\'	2C	'	
\"	22	"	
\?	3F	?	

Le type booléen

- Le type booléen n'est pas prédéfini en C.
- On utilise le type int
  - 0 est faux
  - Toute valeur différente de 0 est vraie, en particulier 1.

Les constantes en C

- Une constante est une valeur qui apparaît littéralement dans le code source d'un programme
  - Exemples : 123, 'A', "Hello", 1.5, ...
- La manière avec laquelle on écrit une constante détermine implicitement son type

Les constantes en C

Catégorie	Représentation	Notation et exemples
Entiers	Décimale Hexa Octale Long unsigned	Habituelle : 123 Commence par **0x** : 0x1F4 → 500 Commence Par **0** : 0377 → 255 Se termine par **l** ou **L** : 123L Se termine par **u** ou **U** : 123U
Réels	A virgule Exponentielle	Habituelle : 12.5 1. .5 Avec **e** ou **E** : 12.4E-5 1e-6 1.5E10 Le type par défaut est double Pour le type float on ajoute **F**/**f** : 1.5F Pour le type long double on ajoute **L**/**l** : 1.5L
Caractères	Caractère Chaine	Entre apostrophes : 'A' '\n' '1' Entre guillemets : "Hello world"

Définition de constantes symboliques

- Définition à l'aide de la directive :
  - **#define NOM Valeur**
- Demande au préprocesseur de remplacer NOM par Valeur dans la suite du fichier source.
- Exemples :
  - **#define PI 3.14**
  - **#define N 100**
  - **#define MIN 0**
  - **#define MAX 100**
  - **#define AVG (MIN + MAX)/2**

Déclaration

- Déclaration de variables :
  - **type identificateur;**
  - **//déclaration et initialisation**
  - **type identificateur = valeur;**
- Déclaration de constantes:
- On utilise le mot clé const
  - **const type identificateur = valeur;**
- Exemple:
  - **const double pi = 3.14**

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques
Affectation	**=**	*variable = expression;* • Ne pas confondre avec ==
arithmétiques	**+ -** *** / %**	• Division entière et réelle : si les deux opérandes sont entières, / produira une division entière (quotient de la division). Exemple : float x; x = 3/2; ➔ x = 1.0 x = 3.0/2; ➔ x = 1.5 • Pas d'opérateur de puissance en C. on utilise la fonction pow(x,y) de math.h
Comparaison	**< <= > >=** **== !=**	*expression1 op expression2* • Le résultat est de type int (pas de type booléen en C): 1 si vrai, et 0 sinon.

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques

Logiques booléens	`&&` (le ET)
	`||` (le OU)
	`!` (le NON)
		• Le résultat est de type int: 1 si vrai, et 0 sinon.
		• L'évaluation d'une expression se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.
		Exemple:
		`int i,j, n;`
		`if (i!=j && i<n && j<n)`
		i<n ne sera évaluée que si i!=j est vraie
		j<n ne sera évaluée que si i!=j et i<n vraies

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques

Logiques bit à bit
 	
(bitwise operators)
	`&`
	`|`
	`^`
	`~`
	`<<`
	`>>`
		Exemple et signification:
		`unsigned char a = 103, b = 41;  //sur 8 bits`
		
		expr binaire déc signification
		a 0110 0111 103 valeur de a
		b 0010 1001 41 valeur de b
		a & b 0010 0001 33 et bit à bit
		a | b 0110 1111 111 ou bit à bit
		a ^ b 0100 1110 78 ou exclusif
		~a 1001 1000 152 complément à 1
		a >>2 0001 1001 25 décalage à droite
		a <<3 0011 1000 56 décalage à gauche

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques

Incrémentation
Décrémentation
	++
	--
		• ++ ajoute 1 à son opérande
		• -- soustrait 1 à son opérande
		• S'utilisent en suffixe (*var++* et *var--*) et en préfixe (*++var* et *--var*).
		Exemple:
		`int x, n;`
		`n = 5;`
		• `x = n++;` incrémentation après affectation
		➔ x = 5 puis n = 6
		• `x = ++n;` incrémentation avant affectation
		➔ n = 6 puis x = 6

Affectation composée	`+=  -=` `*=  /=  %=`	*variable op= expression;* Équivalent à: *variable = variable op expression;*

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques

Opérateur conditionnel ternaire	**? :**	*Condition ? Expression1 : Expression2*
		• Le résultat est Expression1 si la condition est vraie et Expression2 sinon.
		• C'est l'équivalent d'un if – else.
		Exemple :
		`int a, b, max, min;`
		`min = (a<=b) ? a : b;`
		`max = (a>=b) ? a : b;`
		
		`float x;`
		`x = (x>=0) ? x : (-x);`

Les opérateurs en C

Catégorie	Opérateurs	Syntaxe et remarques

Opérateur de conversion explicite de type (cast)	**(type)**	*(type)expression*
		• Permet de modifier **explicitement** le type d'un objet.
		Exemple :
		`int a = 3, b = 2;`
		`float x;`
		`x = a/b;` ➔ `x = 1.0`
		`x = (float)a/b;` ➔ `x = 1.5;`

Opérateur d'adresse	**&**	*&variable* • Appliqué à une variable retourne l'adresse mémoire de cette variable.
Opérateur de taille	**sizeof**	*sizeof(expression)* • expression est un type, une variable, … • Le résultat est le nombre d'octets nécessaires pour stocker l'expression.

Les opérateurs en C

- Ordre de priorité des opérateurs
  - En cas de doute, on utilise les parenthèses

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Unary +, -, and * have higher precedence than the binary forms.

Les entrées sorties standards

Fonctions d'E/S

- Il s'agit des fonctions de la librairie standard **stdio.h** utilisées avec les unités classiques d'entrées / sorties:
  - le clavier et l'écran.
- Pour les utiliser :
  - #include <stdio.h>

Fonction d'affichage

- La fonction **printf**
  - C'est une fonction d'impression formatée
  - ➔ les données sont converties selon le format choisi avant impression.
- Syntaxe:
  - **printf ("chaine de caractères", expression1, …, expressionN);**
  - La chaîne de caractères contient le texte à afficher et les spécifications de format correspondant à chaque expression.

Fonction d'affichage

- Spécificateurs de format pour printf

format	conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	unsigned int	octale non signée
%lo	unsigned long int	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned long int	hexadécimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	chaîne de caractères

Fonction de lecture

- La fonction **scanf**
  - permet de saisir des données au clavier les convertir selon les formats spécifiés puis les stocker en mémoire.
- Syntaxe
  - **scanf ("formats", adresse1, adresse2, ... , adresseN);**

Fonction de lecture

- Format pour scanf

format	type d'objet pointé	représentation de la donnée saisie
%d	int	décimale signée
%hd	short int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%hu	unsigned short int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	int	octale
%ho	short int	octale
%lo	long int	octale
%x	int	hexadécimale
%hx	short int	hexadécimale
%lx	long int	hexadécimale
%f	float	flottante virgule fixe
%lf	double	flottante virgule fixe
%Lf	long double	flottante virgule fixe
%e	float	flottante notation exponentielle
%le	double	flottante notation exponentielle
%Le	long double	flottante notation exponentielle
%g	float	flottante virgule fixe ou notation exponentielle
%lg	double	flottante virgule fixe ou notation exponentielle
%Lg	long double	flottante virgule fixe ou notation exponentielle
%c	char	caractère
%s	char*	chaîne de caractères

Les structures de contrôle

Bloc d'instructions

- Pour délimiter un bloc d'instructions on utilise les { }
  - Deux instructions ou plus.
- Les blocs peuvent être imbriqués:

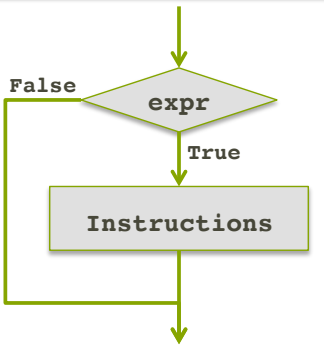
```
Ligne d'entête {
     Instruction 1
     Instruction 2
     ...
     Instruction N
}
```

```
Ligne d'entête {
     Instruction
     Instruction
     Ligne d'entête {
          Instruction
          Instruction
     }
     Instruction
     Instruction
}
```

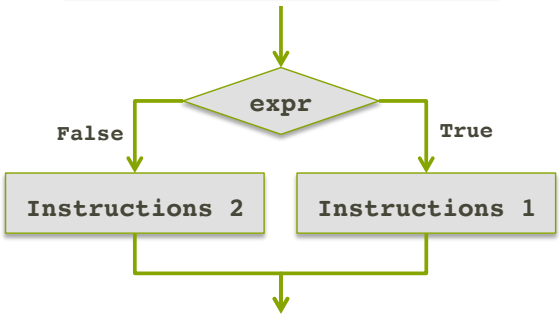
La sélection

- Utilisée pour l'exécution conditionnelle
- Sélection simple :

```
if (expression){
     Bloc d'instructions
}
```



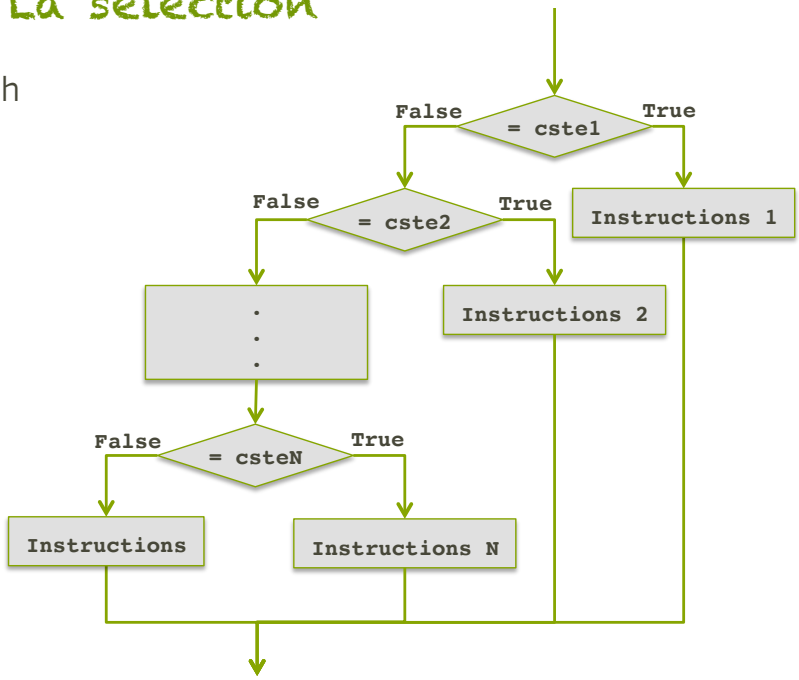

```
if (expression) {
     Bloc d'instructions 1
}
else{
     Bloc d'instructions 2
}
```



La sélection

- Sélection multiple switch

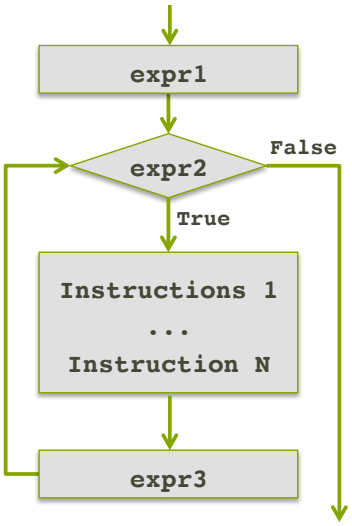
```
switch (expression){
  case constante1 : {
    instructions 1
    break; }
  case constante2 : {
    instructions 2
    break; }
...
  case constanteN : {
    instructions N
    break; }
  default : {
    instructions
    break; }
}
```



La répétition

- La boucle **for**
  - Permet de répéter un bloc d'instructions un nombre prédéfini de fois

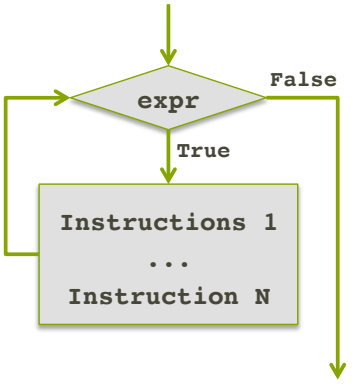
```
for (expr1; expr2; expr3){
     Instruction 1
     ...
     Instruction N
}
```



La répétition

- La boucle **while**
  - Tant que une condition est vraie, on répète l'exécution des instructions
  - On teste, puis on exécute.

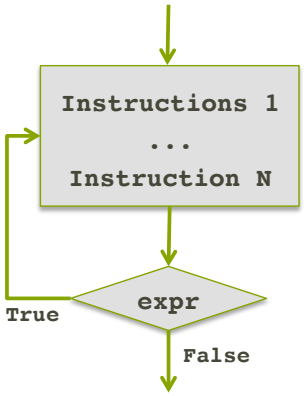
```
while (expression){
     Instruction 1
     ...
     Instruction N
}
```



La répétition

- La boucle do - while
  - Répéter un bloc d'instructions tant que une condition est vraie
  - On exécute, puis on teste.

```
do {
     Instruction 1
     ...
     Instruction N
}while (expression);
```



break et continue

- Utilisées dans une boucle for ou while ou do - while
  - break : permet de sortir de la boucle (ou de switch aussi)
  - continue : permet de passer à l'itération suivante de la boucle

break et continue

```c
#include <stdio.h>

main(){
    int n, p, i, prime;
    printf("Donner un entier : ");
    scanf("%d", &n);

    for (p=2; p<=n; p++){
        prime = 1;
        for (i=2; i<= p/2; i++){
            if (p%i == 0){
                prime = 0;
                break;
            }
        }
        if (prime){
            printf("%d\n", p);
        }
    }
}
```

```c
#include <stdio.h>

main(){
    char *s = "bla bla bla";
    int i;

    for (i = 0; i<11 ; i++){
        if (s[i] == 'l')
            continue;
        printf("%c", s[i]);
    }
}
```