

Langage C – Base I

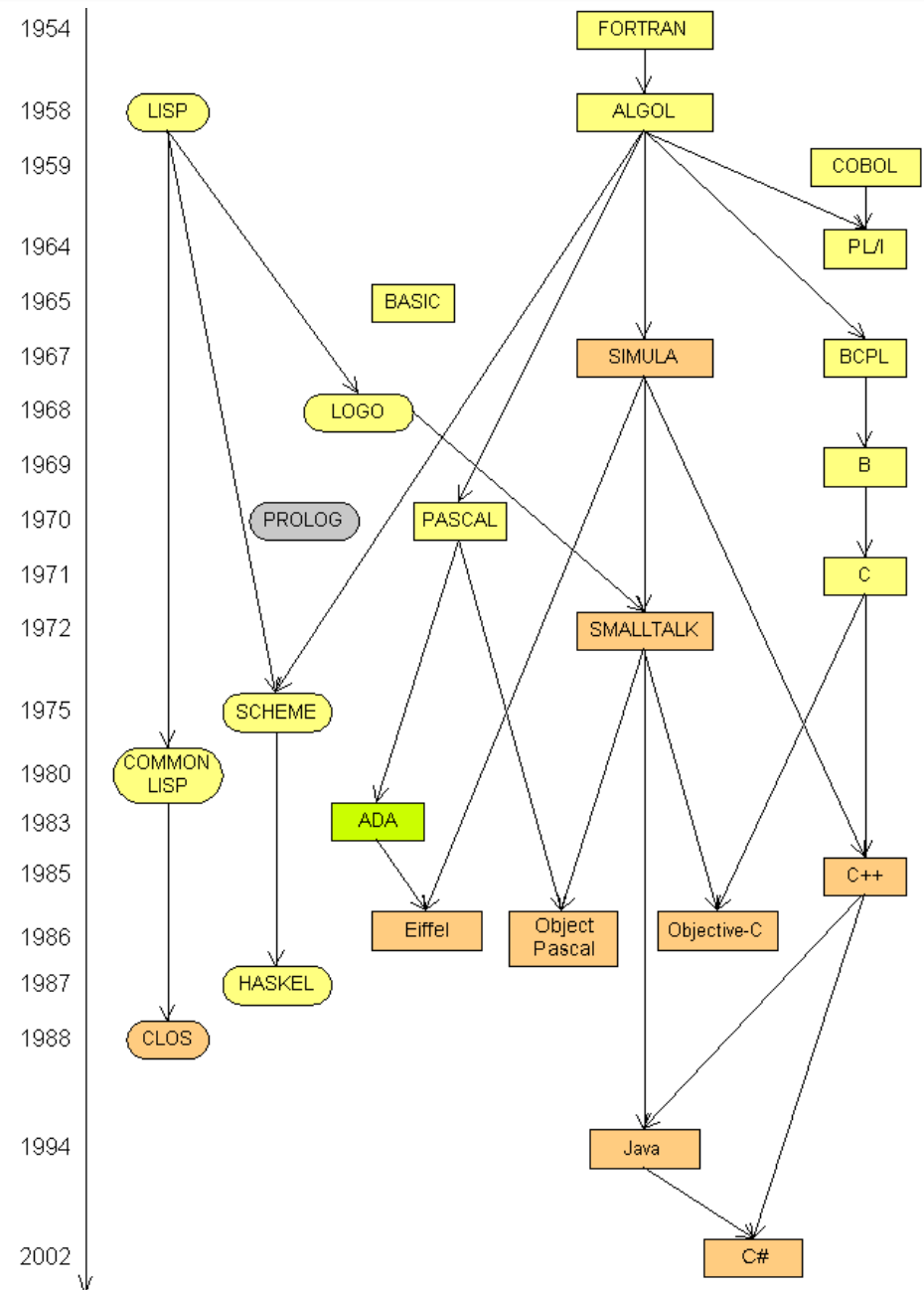
2018

Florent Gluck – Florent.Gluck@hesge.ch

Version 1.7

Historique de C (1)

- Conçu initialement pour la programmation des systèmes d'exploitation (UNIX)
- Créé par Dennis Ritchie à Bell Labs en 1972 dans la continuation de CPL, BCPL et B
- Standardisé entre 1983 et 1988 (ANSI C)
- La syntaxe de C est devenue la base d'autres langages comme C++, Objective-C, Java, GO, C#, etc.
- Révisions plus récentes, notamment C99, puis C11



Historique de C (2)

- Développement de C lié au développement d'UNIX
- UNIX a été initialement développé en assembleur
 - Les instructions assembleur sont de très bas niveau
 - Les instructions assembleur sont spécifiques à l'architecture du processeur
- Pour rendre UNIX portable, un langage de haut niveau était nécessaire
- Le résultat a été la création du langage C, basé sur le langage B



Compaision C ↔ ADA

- C nettement moins typé qu'ADA
- C fait très peu de vérifications: pas de validité des bornes, de variables non initialisées, de pointeurs invalides, etc.
 - ⇒ au programmeur de vérifier son code !
- Les pointeurs sont essentiels en C:
 - Similaire à une référence mais...
 - ... peuvent être utilisé dans des opérations arithmétiques
 - Permet d'utiliser l'adresse des données dans les calculs (pas seulement la valeur)
 - Très puissant, mais la principale source d'erreurs !

Qu'est ce que C ?

- Language typé, compilé, portable et très rapide.
- « bas niveau » pour un langage de « haut niveau ».
- Existe pour presque toutes les architectures.
- **Pas de structures de données haut niveau** comme : chaînes caractères, listes, vecteurs, hash maps, etc.
- Gestion **explicite** de la mémoire : allocation + désallocation
- **Presque aucune validation faite sur les pointeurs !**
- Dangereux : C fait confiance au programmeur (mal) !

Mot-clés de C

Langage très simple, seulement 32 mot-clés (*keywords*) :

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

Types de base

Type	Signification (gcc pour x86-64)
<code>char, unsigned char</code>	Entier signé/non signé 8-bit
<code>short, unsigned short</code>	Entier signé/non signé 16-bit
<code>int, unsigned int</code>	Entier signé/non signé 32-bit
<code>long, unsigned long</code>	Entier signé/non signé 64-bit
<code>float</code>	Nombre à virgule flottante, simple précision
<code>double</code>	Nombre à virgule flottante, double précision

Type booléen ?

- C n'offre pas de type booléen.
- **0 signifie faux et tout ce qui est différent de 0 signifie vrai.**
- Depuis la version C99, la librairie `stdbool.h` met à disposition le type booléen `bool` qui offre les valeurs `true` et `false`.
 - En réalité : entier initialisé à 1 (`true`) ou 0 (`false`).

Quiz : booléens

```
if (42) { ... }  
int x = 100;  
if (x == 4) { ... }  
if (x) { ... }  
while (x--) { ... }  
if (0) { ... }  
if (i = 4) { ... }  
if (i = 0) { ... }
```

**Quelles
expressions
sont vraies ?**

```
#include <stdbool.h>
```

```
bool x = true;  
if (x) { ... }
```

Quiz : booléens

```
if (42) { /* vrai */ }  
int x = 100;  
if (x == 4) { /* faux */ }  
if (x) { /* vrai */ }  
while (x--) { /* répète tant que x est différent de 0 */ }  
if (0) { /* faux */ }  
if (i = 4) { /* vrai */ }  
if (i = 0) { /* faux */ }
```

```
#include <stdbool.h>
```

```
bool x = true;  
if (x) { /* vrai */ }
```

Opérateurs logiques et bit à bit

Opérateur	Exemple	Signification
<	a < b	1 si a < b; 0 autrement
>	a > b	1 si a > b; 0 autrement
<=	a <= b	1 si a <= b; 0 autrement
>=	a >= b	1 si a >= b; 0 autrement
==	a == b	1 si a égal à b; 0 autrement
!=	a != b	1 si a n'est pas égal à b; 0 autrement
&	a & b	ET bit à bit entre a et b
	a b	OU bit à bit entre a et b
^	a ^ b	XOR bit à bit entre a et b
~	~a	NON bit à bit de a
&&	a && b	ET logique entre a et b
	a b	OU logique entre a et b
!	!a	NON logique de a
>>	a >> b	a décalé à droite de b bits
<<	a << b	a décalé à gauche de b bits
++	a++	Incrémentation de a
--	a--	Décrémentement de a

Opérateurs arithmétiques

Opérateur	Dénomination	Effet	Exemple	Résultat (avec $x = 7$)
+	Addition	Additionne deux valeurs	$x + 3$	10
-	Soustraction	Soustrait deux valeurs	$x - 3$	4
*	Multiplication	Multiplie deux valeurs	$x * 3$	21
/	Division	Divise deux valeurs	$x / 3$	2
%	Modulo	Retourne le reste de la division entière	$x \% 3$	1

Opérateurs d'assignation

Opérateur	Effet	Exemple (a = 3)
=	Affecte une valeur (à droite) à une variable (à gauche)	a = 3; // a=3
+=	additionne la valeur (de droite) à la variable (de gauche) et stocke la somme dans la variable	a += 3; // a=6
-=	soustrait la valeur (de droite) à la variable (de gauche) et stocke la différence dans la variable	a -= 3; // a=0
*=	multiplie la valeur (de droite) à la variable (de gauche) et stocke le produit dans la variable	a *= 3; // a=9
/=	divise la valeur (de droite) à la variable (de gauche) et stocke le quotient dans la variable	a /= 3; // a=1
%=	divise la valeur (de droite) à la variable (de gauche) et stocke le reste de la division dans la variable	a %= 2; // a = 1

Quiz : opérateurs

`1 && 0` \Rightarrow ?

`7 && 3` \Rightarrow ?

`7 & 3` \Rightarrow ?

`4 || 3` \Rightarrow ?

`4 | 3` \Rightarrow ?

`!34` \Rightarrow ?

`!0` \Rightarrow ?

Soit `n` un unsigned char initialisé à 127:

`!n` \Rightarrow ?

`~n` \Rightarrow ?

Valeurs ?

Quiz : opérateurs

`1 && 0` \Rightarrow 0

`7 && 3` \Rightarrow 1

`7 & 3` \Rightarrow 3

`4 || 3` \Rightarrow 1

`4 | 3` \Rightarrow 7

`!34` \Rightarrow 0

`!0` \Rightarrow 1

Soit `n` un unsigned char initialisé à 127:

`!n` \Rightarrow 0

`~n` \Rightarrow 128

Structures de contrôle: `if`

- 1 **if** (expression) {
 instructions
}
- 2 **if** (expression) {
 instructions
} **else** {
 instructions
}
- 3 **if** (expression) {
 instructions
} **else if** (expression) {
 instructions
} **else** {
 instructions
}

```
int x = 3;  
char c = 'A';  
  
if (x) {  
    printf("x est vrai");  
    x++  
}  
  
else if (c == 65)  
    printf("lettre A ?");  
else  
    printf("dernier choix");  
x = 0;
```


Structures de contrôle: switch

```
switch (expression) {  
    case constant-expression:  
        instructions  
        break; // optional  
    case constant-expression:  
        instructions  
        break; // optional  
    case constant-expression:  
        instructions  
        break; // optional  
    ...  
    default:  
        instructions;  
}
```

```
int n = 2;  
switch (n) {  
    case 0:  
    case 1:  
        printf("0 ou 1");  
        break;  
    case 2:  
        printf("2");  
        break;  
    default:  
        printf("autre");  
}
```

Structures de contrôle: boucles

- ① **for** (expression1; expression2; expression3) {
 instructions
}
- ② **while** (expression) {
 instructions
}
- ③ **do** {
 instructions
} **while** (expression);

Boucles: exemples (1)

```
int i;  
for (i = 0; i < 10; i++) {  
    printf("%d ", i);  
}
```

```
int i = 0;  
for ( ; i != 1 ; i = rand() % 4) {  
    printf("je continue\n");  
}
```

↓

```
for (int i = 0; i != 1 ; i = rand() % 4) {  
    printf("je continue\n");  
}
```

Syntaxe
C99/C11

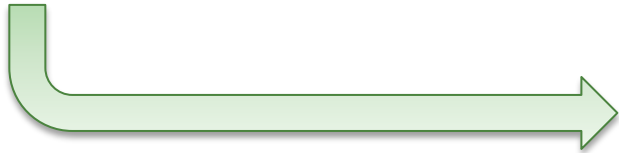
Boucles: exemples (2)

```
int i = 0;
while (i < 10) {
    printf("%d\n", i);
    i++;
}
```

```
char c;
do {
    printf("Pressez 'q' pour quitter...\n");
    scanf("%c", &c);
} while (c != 'q');
```

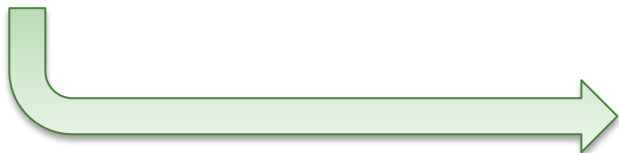
Boucles: break et continue

continue permet de sauter à la prochaine itération



```
for (int i = 0; i < 10; i++) {  
    if (i == 3)  
        continue;  
    printf("%d ", i);  
}
```

break permet de quitter le bloc itératif courant



```
for (int i = 0; i < 10; i++) {  
    if (i == 3)  
        break;  
    printf("%d ", i);  
}
```

Structure d'un programme C

Un programme C est créé à partir:

- Du code **source**:
 - fonctions dans un ou plusieurs fichiers portant l'extension .c
 - une fonction principale (**main**), dans un de ces fichiers
 - optionnellement, des fichiers d'en-tête ou **header** portant l'extension .h
- Optionnellement, des librairies:
 - code binaire externe ainsi que leurs fichiers *header*.

Premier programme

Code du programme main.c

```
#include <stdio.h>    // Inclusion de la librairie standard
                      // pour la gestion des entrées/sorties (I/O)

double global;        // Variable globale

/*  Commentaire
    multi-lignes  */
int main() {
    int local;        // Variable locale à la fonction

    global = 0.42;

    for (local = 0; local < 10; local++) {
        printf("Iteration %d\n", local);
    }

    printf("global = %f\n", global);
    return 0; // Code de retour du programme
}
```

Fonctions

- Une fonction est préfixée par le type de la valeur retournée.
- Une fonction peut avoir de 0 à plusieurs arguments.

```
// Fonction renvoyant la valeur 3 (entier)
int three(void) { // on peut aussi écrire three()
    return 3;
}

// Fonction renvoyant l'addition de x+y (double)
double add(double x, double y) {
    return x+y;
}
```

- Un argument `void` signifie que la fonction n'a pas d'argument
- Une fonction renvoie une valeur à l'aide du mot-clé `return`

Procédures

- Une procédure est une fonction ne renvoyant aucune valeur.
- Ceci est indiqué par le mot-clé `void` en valeur de retour.

```
void compute(int p1, double p2) {  
    ...  
}
```

- Le mot-clé `return` permet de sortir de la procédure à tout moment mais pas de retourner une valeur.

Prototypes de fonctions

- Le compilateur lit toujours le code de haut en bas.
- Le compilateur génère un erreur si une fonction est utilisée **avant** d'avoir été définie.

```
void example(void) {  
    int result = max(5,8);  
}  
  
int max(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

**Ce code
compile-t-il ?**

Prototypes de fonctions

- Le compilateur lit toujours le code de haut en bas.
- Le compilateur génère un erreur si une fonction est utilisée **avant** d'avoir été définie.

```
void example(void) {  
    int result = max(5,8);  
}  
  
int max(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

Non !

Prototypes de fonctions

- Le compilateur lit toujours le code de haut en bas.
- Le compilateur génère un erreur si une fonction est utilisée **avant** d'avoir été définie.

```
void example(void) {  
    int result = max(5,8);  
}  
  
int max(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

Solution(s) ?

Prototypes de fonctions

- Solutions :
 - 1) Déclarer les prototypes de fonctions **avant** leurs utilisations (p.ex. en début de fichier).
 - 2) S'assurer de déclarer les fonctions dans le bon ordre.

```
int max(int a, int b);

void example(void) {
    int result = max(5,8);
}

int max(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

1

```
int max(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}

void example(void) {
    int result = max(5,8);
}
```

2

Fonction `main`

- La fonction `main` est spéciale :
 - Point d'entrée du programme.
 - Retourne le code d'erreur du programme (0 → OK, différent de 0 → erreur).
 - La valeur retournée peut-être lue par le shell qui a exécuté le programme.

```
int main() {  
    ...  
    return 0; // program terminated successfully  
}  
  
int main() {  
    ...  
    return 1; // indicates the program  
              // encountered an error  
}
```

Variables

- **Locale à un bloc:** déclarée à l'intérieur d'un bloc ou d'une fonction; seulement visible dans le bloc où elle est déclarée; la variable est détruite à la sortie du bloc !
- **Globale:** déclarées à l'extérieur d'une fonction; accessible partout et détruite seulement à la sortie du programme.

```
int func() {  
    int x = 2;  // locale, détruite à la sortie de func  
    {  
        int y;  // locale, détruite à la fin du bloc  
        ...  
    }  
    // la variable y n'existe plus ici !  
}  
  
double max;  // globale, détruite à la sortie du programme  
  
int main() {  
    char i;  // locale, détruite à la sortie du programme  
    ...  
}
```

Affichage avec la fonction `printf`

- `printf` est la fonction standard pour afficher du texte :

```
int printf(const char *format, ...);
```

- Plusieurs variantes de la fonction `printf` :
 - `fprintf`, `sprintf`, etc. (cf. manuel avec `man`)
- Nombre d'arguments variable !
- Le premier argument spécifie le format du message à afficher ; ceci inclu, le message, les variables à afficher et leur format.
- Les arguments suivants sont les variables à afficher.

Exemple : printf

```
#include <stdio.h>

int main() {
    int val = 5;
    double x = 0.12871;
    char c = 'a';           // un caractère
    char str[] = "world";   // une chaîne de caractères

    printf("Hello world\n");
    printf("Un entier: %d\n", val);
    printf("Un double arrondi à 2 décimales: %.2g\n", x);
    printf("Un entier et un caractère: %d %c\n", val, c);
    printf("Une chaîne de caractères: %s\n", str);

    return EXIT_SUCCESS;    // equivalent a 0
}
```

Lecture de caractères avec `scanf`

- `scanf` permet de lire des données formatées depuis le clavier.

```
int scanf(const char *format, ...);
```

- Plusieurs variantes de la fonction `scanf` :
 - `fscanf`, `sscanf`, etc. (cf. manuel avec `man`)
- Nombre d'arguments variable !
- Le premier argument spécifie le format du message à lire (format identique à la fonction `printf`).
- Les arguments suivants sont les variables (adresses) où seront stockées les valeurs lues.

Exemple : scanf

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

int main() {
    printf("Enter 3 numbers: ");

    int i, j, k;
    scanf("%d %d %d", &i, &j, &k);
    printf("You entered: %d %d %d\n", i, j, k);

    printf("Enter your name: ");
    char str[512]; // tableau de 512 caractères
    scanf("%s", str);
    printf("Your name is %s\n", str);

    return EXIT_SUCCESS;
}
```

Génération exécutable

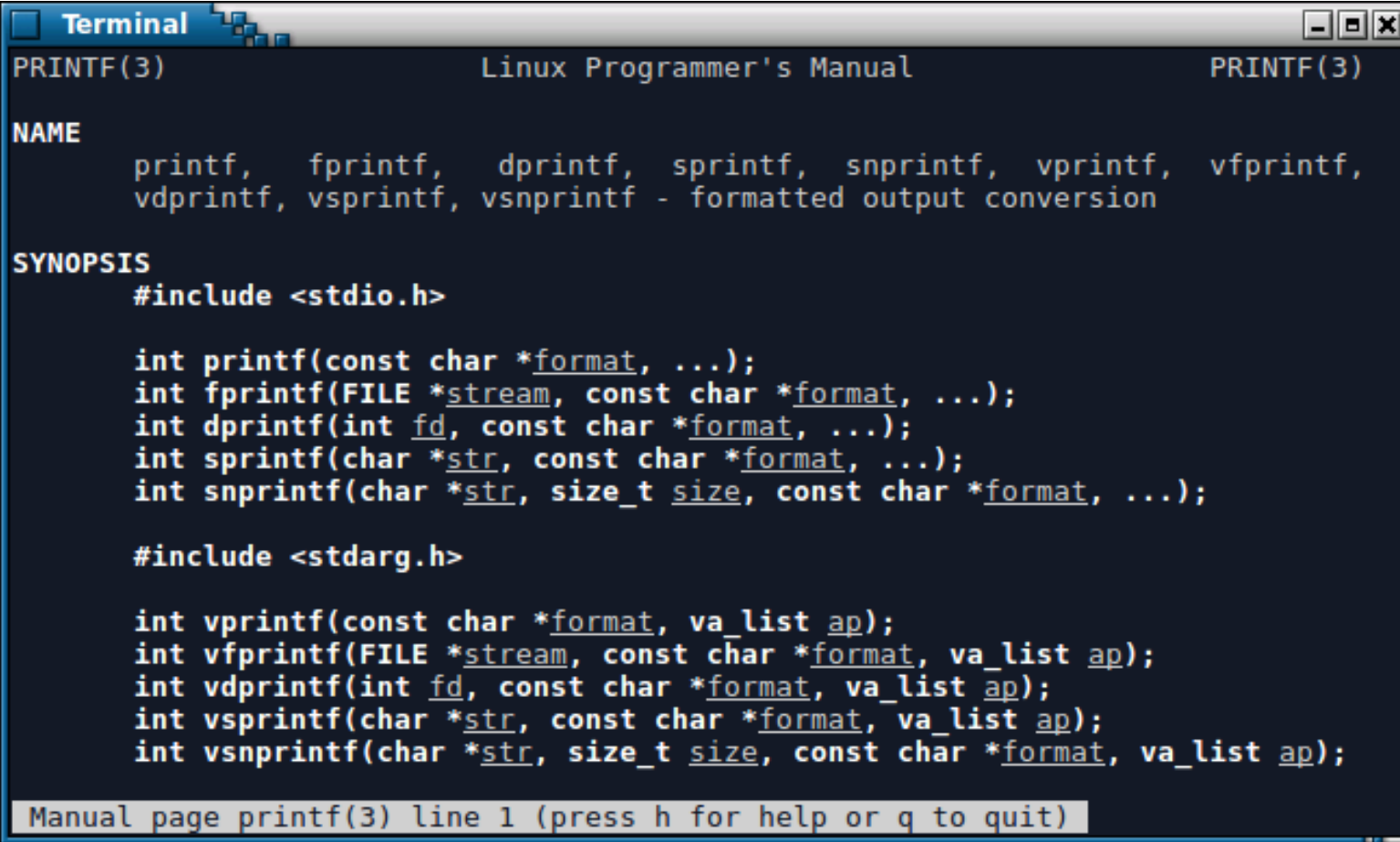
Compilation et édition des liens du code source `prog.c` avec :

```
gcc -std=c11 -Wall -Wextra -fsanitize=address  
-fsanitize=leak -fsanitize=undefined prog.c -o prog
```

- `-std=c11` indique de compiler pour la version 2011 du standard C.
- `-Wall` et `-Wextra` indiquent au compilateur d'alerter le programmeur d'un maximum d'erreurs potentielles.
- `-fsanitize=...` indiquent au compilateur de réaliser des contrôles d'erreurs extensifs à l'exécution (au prix d'un coût en performance).
- Sur Ubuntu 14.04 `-fsanitize=leak` et `-fsanitize=undefined` ne sont pas supportés (`cat /etc/lsb-release` indique la version).
- `-o` définit le fichier exécutable à produire en sortie ; si ce dernier est omis, alors un fichier `a.out` est créé.

Manuel man

- La commande `man` affiche le manuel des fonctions C, commandes systèmes et applications installées.
- Exemple : `"man 3 printf"` (3 spécifie les fonctions C)



```
Terminal
PRINTF(3)                                Linux Programmer's Manual                                PRINTF(3)

NAME
    printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf,
    vdprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
    #include <stdio.h>

    int printf(const char *format, ...);
    int fprintf(FILE *stream, const char *format, ...);
    int dprintf(int fd, const char *format, ...);
    int sprintf(char *str, const char *format, ...);
    int snprintf(char *str, size_t size, const char *format, ...);

    #include <stdarg.h>

    int vprintf(const char *format, va_list ap);
    int vfprintf(FILE *stream, const char *format, va_list ap);
    int vdprintf(int fd, const char *format, va_list ap);
    int vsprintf(char *str, const char *format, va_list ap);
    int vsnprintf(char *str, size_t size, const char *format, va_list ap);

Manual page printf(3) line 1 (press h for help or q to quit)
```

Ressources

The C book

- http://publications.gbdirect.co.uk/c_book/
- Version pdf : http://publications.gbdirect.co.uk/c_book/thecbook.pdf

Le C en 20 heures

- <http://framabook.org/6-le-c-en-20-heures/>

La programmation en C sur wikibooks

- http://fr.wikibooks.org/wiki/Programmation_C

C programming tutorial

- http://www.tutorialspoint.com/cprogramming/cprogramming_pdf_version.htm

Manuel (*man pages*)

- `man stdlib`
- `man stdio...`