

Programmation en C

Premiers éléments du langage C

Alain CROUZIL

`alain.crouzil@irit.fr`

Département d'Informatique (Ddl)

Institut de Recherche en Informatique de Toulouse (IRIT)

Équipe Computational Imaging and Vision (MINDS)

Faculté Sciences et Ingénierie (FSI)

Université Toulouse III – Paul Sabatier (UPS)

Licence Informatique – Licence MIA SHS
2019-2020

Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations
- 3 Entrées-sorties élémentaires
- 4 Constantes
- 5 Opérateurs et expressions
- 6 Instructions de contrôle

Sommaire

1 Structure générale d'un programme

2 Variables, types et déclarations

3 Entrées-sorties élémentaires

4 Constantes

5 Opérateurs et expressions

6 Instructions de contrôle

Structure générale d'un programme (1/3)

Format d'un programme

- Texte structuré en lignes
- Mise en page libre
- Indentation nécessaire pour la lisibilité

Commentaires

```
/* commentaire */
```

Un commentaire peut :

- être placé partout où peut apparaître un espace ;
- occuper plusieurs lignes :

```
/* Ceci est  
un commentaire */
```

Commentaire de fin de ligne (depuis C99) :

```
// Commentaire de fin de ligne
```

Structure générale d'un programme (2/3)

Ensemble de fonctions

Programme :

- ensemble de fonctions disjointes
- une porte le nom de `main` : fonction principale (\supset 1^{ère} instruction du programme)

Fonctions :

- structurées en blocs
- regroupées en fichiers

FICHER

Déclarations
Définition fonction 1
Définition fonction 2
⋮

FONCTION

En-tête
Bloc

BLOC

{
Déclarations
Instruction 1
Instruction 2
⋮
}

Structure générale d'un programme (3/3)

Hiérarchie de blocs

Un bloc peut contenir des blocs.

```
1  { /* Début du bloc de niveau 1 */  
2    int c;  
3    c=getchar();  
4    while (c!=EOF)  
5      { /* Début du bloc de niveau 2 */  
6        putchar(c);  
7        c=getchar();  
8      } /* Fin du bloc de niveau 2 */  
9  } /* Fin du bloc de niveau 1 */
```

Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations**
- 3 Entrées-sorties élémentaires
- 4 Constantes
- 5 Opérateurs et expressions
- 6 Instructions de contrôle

Variables et types (1/6)

Identificateurs

- Suite d'éléments parmi :
a,b,...,z,A,B,...,Z,_,0,1,...9
- Majuscules et minuscules différenciées.
- Premier caractère : pas un chiffre.
- Mots-clés du langage (réservés) :

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

Variables et types (2/6)

Déclaration



Toutes les variables utilisées dans un programme C doivent avoir été déclarées avant leur utilisation.

Instruction de déclaration (association d'une liste d'identificateurs de variables à un type) :

$$\text{type } \textit{ident}_1, \textit{ident}_2, \dots, \textit{ident}_n;$$

Types caractères

Caractère	char
Petit entier	unsigned char ([0, 255]) ou signed char ([−127, +127])

char alpha,beta;
unsigned char pixel;
char c;
 c = ' : ' ;

Représentation mémoire de la variable `c` :

0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

c'est-à-dire : $(+60)_{10}$.

Variables et types (3/6)

Types entiers

Les types entiers se distinguent par :

- un paramètre de signe (facultatif) : `signed` **ou** `unsigned`
- un paramètre de taille (facultatif) : `short` **ou** `long`

Entier court signé	<code>signed short int</code> ou <code>short</code>
Entier court non signé	<code>unsigned short int</code> ou <code>unsigned short</code>
Entier signé	<code>signed int</code> ou <code>int</code>
Entier non signé	<code>unsigned int</code> ou <code>unsigned</code>
Entier long signé	<code>signed long int</code> ou <code>long</code>
Entier long non signé	<code>unsigned long int</code> ou <code>unsigned long</code>

$taille(\text{entiers courts}) \leq taille(\text{entiers}) \leq taille(\text{entiers longs}).$

Variables et types (4/6)

Types flottants

Les types flottants se distinguent par :

- le domaine ;
- la précision : une précision égale à n signifie que tout entier d'au plus n chiffres s'exprime sans erreur en flottant.

Flottant	float
Flottant double précision	double
Flottant long double précision	long double

Variables et types (5/6)

Taille des types de base

Dénomination	Types	Taille minimale (octets)	Domaine minimal
Caractère non signé	unsigned char char (*)	1 exactement	[0..255] exactement
Caractère signé	signed char char (*)	1 exactement	[−127.. +127]
Entier court signé	short short int signed short signed short int	2	[−32767.. +32767]
Entier court non signé	unsigned short unsigned short int	2	[0..65535]
Entier signé	int signed int signed	2	[−32767.. +32767]
Entier non signé	unsigned int unsigned	2	[0..65535]
Entier long signé	long long int signed long signed long int	4	[−2147483647.. +2147483647]
Entier long non signé	unsigned long unsigned long int	4	[0..4294967295]
Flottant	float	—	$[-10^{+37} .. -10^{-37}] + [10^{-37} .. 10^{+37}]$ Précision minimale : 6
Flottant double précision	double	—	$[-10^{+37} .. -10^{-37}] + [10^{-37} .. 10^{+37}]$ Précision minimale : 10
Flottant long double précision	long double	—	$[-10^{+37} .. -10^{-37}] + [10^{-37} .. 10^{+37}]$ Précision minimale : 10

(*) Cela dépend de l'implémentation.

Variables et types (6/6)

Initialisation



Après sa déclaration, une variable a une valeur indéterminée.

Possibilité d'initialiser une variable lors de la déclaration :

```
int Somme=0;
```

```
int delai=12*60;
```

```
int a,b=0; /* attention : a n'est pas initialisée */
```

```
char c=' X ';
```

À vos boîtiers !

2



Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations
- 3 Entrées-sorties élémentaires**
- 4 Constantes
- 5 Opérateurs et expressions
- 6 Instructions de contrôle

Entrées-sorties élémentaires (1/4)

Pour pouvoir faire des exercices rapidement

- Nous n'allons voir dans cette partie que quelques possibilités simples d'entrées-sorties.
- Des explications plus détaillées et d'autres possibilités seront données plus tard et se trouvent dans le support complémentaire.

Bibliothèque standard

Pour pouvoir utiliser les fonctions d'entrées-sorties, il faut placer en début de programme :

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


Entrées-sorties élémentaires (2/4)

Lecture au clavier

La fonction `scanf` permet de faire des lectures formatées.
Supposons les déclarations suivantes :

```
char c;  
int i;  
float x;  
double y;
```

- Lecture d'un caractère : `scanf("%c",&c);`
- Lecture d'un entier : `scanf("%d",&i);`
- Lecture d'un réel : `scanf("%f",&x); scanf("%lf",&y);`

Entrées-sorties élémentaires (3/4)

Affichage à l'écran

La fonction `printf` permet d'afficher à l'écran du texte et des valeurs d'expressions.

```
printf(format, expr1, expr2, ... );
```

- *format* est une chaîne de caractères, délimitée par des guillemets ("), contenant du texte et des spécificateurs de format commençant par %.
- Le texte normal est affiché tel quel, mais les spécificateurs de format sont remplacés par les valeurs des expressions correspondantes.
- Le premier spécificateur de format correspond à la première expression, le deuxième à la deuxième expression, etc.

Principaux spécificateurs de format :

- %c pour un caractère
- %d pour un entier
- %f pour un réel

Entrées-sorties élémentaires (4/4)

Affichage à l'écran (suite)


```
int i; char car;  
i=5; car=' a';  
printf("Nombre_:_%d_;_caractere_:_%c\n",i,car);
```

provoque à l'écran l'affichage suivant :

Nombre : 5 ; caractere : a

Remarques :

- Le caractère `\n` est le caractère « nouvelle ligne ».

-  La fonction `printf` envoie la chaîne de caractères interprétée, non pas directement à l'écran, mais sur le « tampon de sortie ». Pour forcer l'affichage du contenu du tampon de sortie à l'écran, il existe deux possibilités :
 - soit on termine la chaîne de caractères *format* par un caractère `\n` ;
 - soit on place l'instruction `fflush(stdout)` ; juste après l'appel de la fonction `printf`, afin de vider le tampon de sortie.

Démo



Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations
- 3 Entrées-sorties élémentaires
- 4 Constantes**
- 5 Opérateurs et expressions
- 6 Instructions de contrôle

Constantes (1/4)

Constantes entières

On peut utiliser la représentation en base 10, 8 ou 16 :

- en décimal (base 10) : 14, -5, 65535
- en octal (base 8), on fait précéder le nombre du chiffre 0 : 016, 0177777
(attention : $016 \neq 16$)
- en hexadécimal (base 16), on fait précéder le nombre de 0x (ou 0X) : 0xe, 0xFFFF

Constantes réelles

On peut utiliser :

- la notation décimale : 3.141
- la notation « scientifique » :
0.3141e+1 ou 0.3141E+1
0.03141e+2 ou 0.03141E+2
31.41e-1 ou 31.41E-1

Constantes (2/4)

Constantes caractères

- Caractères « imprimables » : 'A', '4', '0'
- Caractères disposant d'une notation symbolique :

Notation	Dénomination	Notation	Dénomination	Notation	Dénomination
'\n'	saut de ligne (<i>new line</i>)	'\f'	saut de page	'\?'	point d'interrogation
'\t'	tabulation	'\\'	anti-slash	'\a'	bip
'\b'	retour arrière (<i>backspace</i>)	'\''	apostrophe	'\v'	tabulation verticale
'\r'	retour chariot	'\"'	guillemets		

- Désignation d'un caractère par son code (anti-slash suivi du code ASCII du caractère en octal ou en hexadécimal) : '\x41' ('A'), '\101' ('A'), '\0' (caractère « nul »).

Constantes (3/4)

Constantes chaînes de caractères

Elles sont placées entre guillemets ("). Elles peuvent contenir des caractères en notation imprimable, octale, hexadécimale ou symbolique :

- "AZERTY" est une chaîne de 6 caractères
- "a3(\n" est une chaîne de 4 caractères
- "a3(\\n" est une chaîne de 5 caractères
- "bonjour\nmonsieur" est une chaîne de 16 caractères
- "ab\"cd\" " est une chaîne de 6 caractères



En mémoire, elles sont complétées par le caractère '`\0`' :

"AZERTY"

'A'
'Z'
'E'
'R'
'T'
'Y'
'\0'

"a"

'a'
'\0'

'a'

'a'

" "

'\0'

Constantes (4/4)

Définition de constantes symboliques

Préprocesseur :

- programme qui effectue un pré-traitement lors de la compilation ;
- supprime les commentaires ;
- traite les directives (lignes commençant par #) ;
- envoie le programme modifié au compilateur.

Forme simple des directives de substitution symbolique :

`#define symbole équivalent`

remplace dans la suite les occurrences de *symbole* par son *équivalent*, sauf :

- dans les lignes commençant par le caractère #,
- dans les constantes chaînes de caractères (situées entre guillemets),
- au milieu d'un identificateur (si *symbole* est précédé ou suivi d'un des caractères autorisés pour les identificateurs de variables).

Exemple : **#define** LONGUEUR 100

Remarque : Avec `gcc`, la commande `gcc -E -P prog.c` affiche à l'écran le programme `prog.c` après le passage du préprocesseur.



Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations
- 3 Entrées-sorties élémentaires
- 4 Constantes
- 5 Opérateurs et expressions**
- 6 Instructions de contrôle

Opérateurs et expressions (1/8)

Expressions

- Une *expression* est constituée d'*opérandes* et d'*opérateurs*.
- Toute donnée placée en mémoire centrale est considérée comme une valeur numérique.
 - ⇒ Un caractère peut être utilisé dans une expression arithmétique. Le code du caractère est considéré comme un `int`.
 - ⇒ Une expression logique (« booléenne ») est considérée comme un entier valant 0 (faux) ou 1 (vrai). Par exemple, $(7 > 3)$ vaut 1.
 - ⇒ Des conversions implicites de types peuvent avoir lieu. Une « hiérarchie » entre les types est définie :

`char` → `int` → `long` → `float` → `double` → `long double`

Si l'on déclare :

int n; **double** x;

dans l'expression `n+x`, la valeur de `n` est convertie en `double` et le résultat est de type `double`.

Opérateurs et expressions (2/8)

Opérateurs d'affectation

- Affectation simple : =

int i;

i=12; /* la variable i reçoit la valeur 12 */

- Incrémentation et décrémentation : ++ --

Désignation	Expression	Effet	Exemple
Post-incrémentation	<i>variable</i> ++	La valeur de <i>variable</i> est augmentée de 1. L'expression vaut l'ancienne valeur de <i>variable</i> .	i=1; a=i++; a vaut 1 et i vaut 2
Pré-incrémentation	++ <i>variable</i>	La valeur de <i>variable</i> est augmentée de 1. L'expression vaut la nouvelle valeur de <i>variable</i> .	i=1; a=++i; a vaut 2 et i vaut 2
Post-décrémentation	<i>variable</i> --	La valeur de <i>variable</i> est diminuée de 1. L'expression vaut l'ancienne valeur de <i>variable</i> .	i=1; a=i--; a vaut 1 et i vaut 0
Pré-décrémentation	-- <i>variable</i>	La valeur de <i>variable</i> est diminuée de 1. L'expression vaut la nouvelle valeur de <i>variable</i> .	i=1; a=--i; a vaut 0 et i vaut 0

Opérateurs et expressions (3/8)

Opérateurs d'affectation (suite)

- Affectations multiples : l'évaluation est effectuée de la droite vers la gauche.

int i,j,k;

i=j=k=0; \Leftrightarrow k=0; j=0; i=0;

- Affectations élargies : on peut condenser

variable = variable opérateur expression

en

variable opérateur= expression

x+=5; \Leftrightarrow x=x+5;

x*=x; \Leftrightarrow x=x*x;

Opérateurs et expressions (4/8)

Opérateurs

- Opérateurs arithmétiques :

+	-	*	/	%
---	---	---	---	---



/ : division entière si les deux opérandes sont entiers, division réelle sinon.

% : reste de la division entière

- Opérateurs relationnels :

==	!=	>	<	>=	<=
----	----	---	---	----	----

- Opérateurs logiques :

! (NON logique)	&& (ET logique)	(OU logique)
-----------------	-----------------	--------------



L'évaluation des opérateurs logiques est faite de gauche à droite et s'arrête dès que la décision peut être prise.

Opérateurs et expressions (5/8)

Expressions logiques

- Une expression logique vaut 0 si elle est fausse ou 1 si elle est vraie.
- Une expression est considérée comme fausse si elle vaut 0, vraie sinon ($\neq 0$).

`x=7;`

`! ((x!=2) && (5>=0))` vaut 0 (faux)

`(-5) && (x!=2)` vaut 1 (vrai)

`(24!=1) || (5<0)` vaut 1 (vrai)

`(7>2) && 5` vaut 1 (vrai)

`!3` vaut 0 (faux)

À vos boîtiers !

3



Opérateurs et expressions (6/8)

Opérateur conditionnel

$$expression_1 \text{ ? } expression_2 \text{ : } expression_3$$
$$\text{vaut } \begin{cases} expression_2 \text{ si } expression_1 \neq 0 \text{ (vraie)} \\ expression_3 \text{ si } expression_1 \text{ vaut } 0 \text{ (fausse)} \end{cases}$$

`x=n>0?n:-n; /*valeur absolue */`

`x=a>b?a:b; /*max */`

Opérateurs et expressions (7/8)

Conversions forcées de type

- Conversion d'une expression dans un type donné grâce à l'opérateur de *cast* :
(*type*)

```
int a=7,b=2;
```

- L'expression `(double) (a/b)` est de type `double` et vaut 3.0.
- L'expression `(double) a/b` est aussi de type `double` et vaut 3.5 (la valeur de `a` est d'abord convertie en `double`, puis la division est effectuée en `double`).

- Conversion par affectation :

```
int a=7,b=2; double x;
```

```
x=a/b; /*x vaut 3.0 */
```

```
a=x/b; /*a vaut 1 */
```



Il faut être prudent quand on effectue des conversions dans « le mauvais sens » de la hiérarchie des types car la valeur peut ne pas être représentable.

Opérateurs et expressions (8/8)

Priorités des opérateurs

- Un tableau donné dans le support complémentaire indique les priorités par ordre décroissant des différents opérateurs ainsi que les sens d'évaluation (gauche \rightarrow droite ou gauche \leftarrow droite).
- Dans le doute, utilisez des parenthèses !

Sommaire

- 1 Structure générale d'un programme
- 2 Variables, types et déclarations
- 3 Entrées-sorties élémentaires
- 4 Constantes
- 5 Opérateurs et expressions
- 6 Instructions de contrôle**

Instructions de contrôle (1/5)

Point-virgule

Le point-virgule (;) est le *terminateur d'instruction*.

Conventions

Dans la suite :

- *inst* désigne une instruction simple ou un bloc d'instructions ;
- *expr* désigne une expression ;
- les crochets ([]) entourent une partie facultative.

Instructions de contrôle (2/5)

Choix

```
if (expression) inst1 [else inst2]
```

```
if (x==3) printf("OK\n");
```

```
if (a>b) m=a; else m=b;
```

Choix en cascade :

```
if (prix<100) tranche=1;
```

```
else if (prix<200) tranche=2;
```

```
else if (prix<300) tranche=3;
```

```
else tranche=4;
```



Attention aux imbrications : un `else` se rapporte au dernier `if` rencontré dans le même bloc auquel un `else` n'a pas encore été attribué.

```
int a=1,b=3,c=2;
```

```
if (a<b)
```

```
    if (b<c) printf("a<b<c\n");
```

```
else printf("a>=b\n");
```

provoque l'affichage de : `a>=b` !

```
int a=1,b=3,c=2;
```

```
if (a<b)
```

```
{
```

```
    if (b<c) printf("a<b<c\n");
```

```
}
```

```
else printf("a>=b\n");
```

ne va provoquer aucun affichage.

Instructions de contrôle (3/5)

Choix (suite)



Attention aux opérateurs :

if ($r=n\%d$) ... \iff si r (qui reçoit n modulo d) est différent de 0 ...

if ($r==n\%d$) ... \iff si r est égal à n modulo d ...

Instructions de contrôle (4/5)

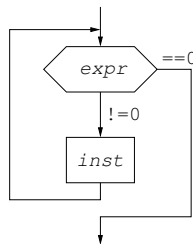
Répétition : boucle `while`

```
while (expr) inst
```

Tant que *expr* est différente de 0, *inst* est exécutée (si *expr* vaut 0 au départ, *inst* n'est pas exécutée).

```
a=0;
while (a<6)
{
    printf("%d_",a);
    a+=2;
}
```

provoque l'affichage de 0 2 4 . Mais si *a* est initialisée à 6, on n'obtient aucun affichage.



Instructions de contrôle (5/5)

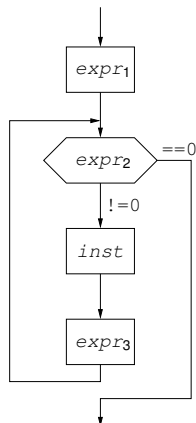
Répétition : boucle `for`

`for ([expr1]; [expr2]; [expr3]) inst`

La boucle `for` peut se réécrire à l'aide de la boucle `while` :

```
expr1;  
while (expr2)  
{  
    inst  
    expr3;  
}
```

for (*a*=0;*a*<6;*a*+=2)
 printf("%d_",*a*);
provoque l'affichage de 0 2 4 .



À vos boîtiers !

4

