

BASES DU LANGAGE C

2012-10-01

© Emmanuel Chieze
Département d'informatique, UQAM
INF3135

Plan

- **Généralités sur le C**
- Programmes en C
- Variables et constantes
- Structures de contrôle
- Opérateurs
- Exercices

Présentation du langage C

3

- Langage de programmation procédurale
- Langage généraliste
- Langage de "bas niveau" :
  - pas d'opérations pour E/S
  - pas d'opérations pour gérer des données complexes
  - mais utilisation de bibliothèques de fonctions pour ces opérations
- Langage compilé
- Langage très efficace

Présentation du langage C

4

- Langage étroitement lié à UNIX à l'origine
  - développé dans les années 70 par Ritchie (Bell Labs)
  - pour le système d'exploitation UNIX : 95 % de UNIX écrit en C
- Langage indépendant de toute plateforme
- 1989 : Norme C ANSI
- Permet la programmation structurée
- (et permet aussi la programmation spaghetti)

Présentation du langage C

5

- Permet une approche modulaire de la programmation
  - développement de bibliothèques de fonctions compilées indépendamment des programmes principaux
  - réutilisation des bibliothèques de fonctions
- Regroupement des opérations
  - en blocs d'opérations
  - à l'intérieur de fonctions

Plan

6

- Généralités sur le C
- **Programmes en C**
- Variables et constantes
- Structures de contrôle
- Opérateurs
- Exercices

Structure d'un programme C

7

1. Directives au préprocesseur
   - inclusion de bibliothèques de fonctions
   - définition de constantes
2. Déclaration obligatoire des fonctions
3. Code de la fonction principale main()
4. Code des fonctions déclarées

Structure d'un programme C

8

- Blocs d'opérations définis par { et }
- Commentaires définis par /* et */
  - multilignes le cas échéant
- Chaque instruction se termine par ;
- cf. Exemple1.1.c

Écriture d'un programme C

9

- Pour la lisibilité des programmes
  - Une seule instruction par ligne
  - Laisser une ligne blanche entre les déclarations et les instructions
  - } est seule sur une ligne et est indentée avec le début du bloc
    - sauf pour do … while
  - Utiliser l'indentation
    - même indentation pour les différentes instructions d'un même bloc
  - Commenter les programmes

Exécution d'un programme C

10

- Écriture du programme source au moyen d'un éditeur de texte
  - vi, emacs, Notepad …
  - le nom du source se termine par .c
- Compilation du programme source au moyen d'un compilateur
  - Vérifie la syntaxe du code
  - gcc –W –Wall -o <fichier exécutable> <fichier source>
  - Exécutable par défaut avec gcc : a.out
- Exécution du fichier exécutable

Plan

11

- Généralités sur le C
- Programmes en C
- **Variables et constantes**
- Structures de contrôle
- Opérateurs
- Exercices

Identificateurs en C

12

- Identificateur = nom des fonctions et des variables
  - commencent par une lettre
  - sont composés de lettres (incluant _) et de chiffres
  - sont sensibles à la casse
  - les 31 premiers caractères sont significatifs (ANSI)

Identificateurs en C

13

- Utiliser des identificateurs significatifs
- Utiliser des identificateurs commençant par une lettre minuscule pour variables et fonctions
- Utiliser des identificateurs en majuscules pour le préprocesseur
- Utiliser des identificateurs commençant par _ pour le compilateur

Identificateurs en C

- Un identificateur ne peut être un mot-clé réservé du C

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

Types de données entiers

Type	Taille min. (octets)	Valeur min.	Valeur max.
char	1	-128	127
signed char	1	-128	127
unsigned char	1	0	255
short	2	-32768	32767
unsigned short	2	0	65535
int	Entre 2 et 4 octets 4 sur Arabica	-2147483648	2147483647
unsigned	cf. int	0	4292967295
long	4	-2147483648	2147483647
unsigned long	4	0	4292967295

Types de données flottants

Type	Taille min. (Octets)	Valeur min.	Valeur max.
float	4	1.17E-38	±3.4E38
double	8	2.22E-308	±1.7E308
long double	16	3.36E-4932	±1.1E4932

Autres types de données atomiques

17

- Type vide : void
  - sert à définir le type d'une fonction ne retournant rien
- Aucun type booléen
  - toute valeur entière peut être considérée comme booléenne
  - 0 équivaut à Faux
  - Toute autre valeur équivaut à Vrai

Déclaration des variables

18

- Variables automatiques :
  - Doivent être déclarées avant leur utilisation, en début de bloc
  - Locales au bloc où elles sont déclarées
  - Visibles jusqu'à la fin du bloc
  - Déclaration et initialisation peuvent être combinées
  - Valeur fantaisiste si absence d'initialisation

```
char c = 'e';

/* a n'est pas initialisé, b l'est à 4 */
int a, b=4;

/* initialisation avec la valeur d'une expression */
unsigned d = fact(10);
```

Visibilité des variables

19

- Qu'affiche le programme Exemple1.2.c ?
  - i vaut 0, j vaut 100
  - i vaut 5, j vaut 100
  - i vaut 0, j vaut 10
  - i vaut 0, j vaut 100

Constantes

20

- Peuvent être définies comme des symboles du préprocesseur

```
#define PI 3.14159
```

- Ou peuvent être définies comme variables avec la spécification que leur valeur est constante

```
const float pi = 3.14159;
```

- Ou par la commande enum

```
enum {
  DEBUT       = 1,
  FIN         = 10
};
```

Constantes

21

- Notation décimale par défaut : 48
  - suffixe u ou U pour indiquer une constante non signée : 348u
  - suffixe l ou L pour indiquer une constante longue
- Notation octale : 060
- Notation hexadécimale : 0X30 /0x30
- Notation sous la forme d'un caractère : '0'

```
char i = 48, j = 060, k = 0X30, l = '0';
printf("%d %d %d %d\n", i, j, k, l);
```

affiche

```
48 48 48 48
```


Constantes

22

- Quelques caractères spéciaux
  - `\n` représente NL
  - `\t` TAB
  - `\\` \
  - `\'` '
  - `\"` "

Constantes

23

- Utilité de déclarer des valeurs fixes comme constantes plutôt que des les utiliser telles quelles dans les programmes ?
- Utilité de déclarer ces valeurs comme constantes plutôt que comme simples variables ?

Plan

24

- Généralités sur le C
- Programmes en C
- Variables et constantes
- **Structures de contrôle**
- Opérateurs
- Exercices

Structures de contrôle

25

- Instruction for

```
for (<initialisation> ; <condition de continuation> ;
  <incrémentation>)
  <instruction>
```

  - <initialisation> est évaluée une fois, au début de l'exécution de la boucle
  - <condition> est évaluée à chaque passage, avant d'exécuter <instruction>
  - <incrémentation> est évaluée à chaque passage, après avoir exécuté <instruction>

Structures de contrôle

26

- Que fait le code suivant ?

```
unsigned i, f=1;
for (i=1; i<=n ; i++)
  f = f * i;
```

Structures de contrôle

27

- Instruction if … then … else
  - `if (<condition>)`
    `<instruction>;`
  - `if (<condition>)`
    `<instruction>;`
    `else`
    `<instruction>;`
  - `if (<condition>)`
    `<instruction>;`
    `else if (<condition>)`
    `<instruction>;`
    `else`
    `<instruction>;`

Structures de contrôle

28

- Else est associé au dernier if sans else qui le précède
- Comparer :

```
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

```
if (n > 0) {
      if (a > b)
            z = a;
}
else
      z = b;
```

Structures de contrôle

29

- Instruction switch (aiguillage)

```
switch (<variable>) {
  case <valeur> : <instruction>;
  case <valeur> : <instruction>;
  default : <instruction>;
}
```

  - Default : optionnel
  - À partir du moment où la valeur de la variable est trouvée, toutes les instructions jusqu'à la fin du switch sont exécutées
    - utiliser un break pour sortir du switch
    - Ordre des cas sans importance si utilisation du break pour chaque cas
    - On peut n'écrire qu'une fois les instructions communes à plusieurs cas (mais attention à ne pas changer l'ordre des cas).

Structures de contrôle

30

- Instruction while : la condition est testée avant que les instructions ne soient exécutées

```
while (<condition>)
  <instruction>;
```

- Instruction do-while : la condition est testée après que les instructions ont été exécutées

```
do
  <instruction>;
while (<condition>);
```

Structures de contrôle

31

- Instruction break : permet de sortir prématurément d'une boucle while, do-while, for
- Instruction continue : permet d'aller directement en fin de boucle (pour démarrer prématurément l'itération suivante)

Plan

32

- Généralités sur le C
- Programmes en C
- Variables et constantes
- Structures de contrôle
- **Opérateurs**
- Exercices

Opérateurs arithmétiques

Opérateur	Opération	Utilisation
+	addition	x + y
-	soustraction	x - y
-	opposé	-x
*	multiplication	x * y
/	division	x / y
%	modulo	x % y

Opérateurs arithmétiques

- Entiers négatifs
  - Représentés par le complément à 2 : on inverse les bits et ajoute 1
- Arithmétique sur les entiers : circulaire
  - Il n'y a jamais débordement
- Exemple :

```
char c = 255, c1 = c+1;
unsigned char d = -1, d1 = d+1;
printf("%d %d %d %d\n", c, d, c1, d1);
```

affiche

```
-1 255 0 0
```

Opérateurs arithmétiques

35

- Attention aux conversions implicites entre types signés et non-signés
  - -1 d'un type signé correspond à la valeur max d'un type non signé

```
char x=-1, y = 20, v;
unsigned char z = 254;
unsigned short t;
unsigned short u;

t=x;
u=y;
v=z;  /* t=65535, u=20, v=-2 */
```


Conversion de types

36

- Conversion automatique dans une opération :
  - si l'un des opérandes est long double, le résultat est long double,
  - sinon, si l'un des opérandes est double, le résultat est double,
  - sinon, si l'un des opérandes est float, le résultat est float,
  - sinon, promotion des entiers : la conversion se fait vers le type int (ou unsigned, si une opération est unsigned)
- Évitez de mélanger entiers signés et non signés au sein d'une même opération

Opérateurs de comparaison

Opérateur	Opération	Utilisation
==	égalité	x == y
!=	inégalité	x != y
>	supérieur	x > y
>=	supérieur ou égal	x >= y
<	inférieur	x < y
<=	inférieur ou égal	x <= y

Opérateurs logiques

Opérateur	Opération	Utilisation
!	non	! x
&&	et	x && y
||	ou	x || y

- Évaluation paresseuse de && et ||

Opérateurs d'affectation

39

- =, +=, -=, *=, /=, %=

```
int x = 1, y, z, t;
t = y = x ;    /* équivaut à t = (y = x) */
x *= y + 1; /* équivaut à x = x * (y + 1) */
```

- Incrémentation / Décrémentation : ++ / --

```
int x = 1, y, z ;
y = x++ ; /* y = 1, x = 2 : post-incrémentation */
z = ++x ; /* z = 3, x = 3 : pré-incrémentation */
```

- Nécessitent une expression spécifiant une zone de stockage à gauche

```
(y+1) *= x ; /* INVALIDE */
```


Opérateurs divers

40

- Opérateur ternaire ? :

(<condition> ? <expression si vrai> : <expression si faux>)

```
int x = 1, y, z ;
/* Que valent y et z ? */
y = (x-- == 0 ? 1 : 2) ;
z = (++x == 1 ? 1 : 2) ;
```

Opérateurs divers

41

- Opérateur de séquençage ,
  - résultat correspond au type et à la valeur de l'expression à droite
  - utile pour ses effets de bord

```
z = (t++, x-1) ;
```

- Autres opérateurs : opérateurs bit à bit

Conversion de types

42

- Conversion explicite ('cast') :

```
unsigned char x = 255;
printf("%d\n", x) ; /* affiche 255 */
printf("%d\n", (signed char)x) ;  /*
   affiche -1 */

int x=3,y=4;
printf("%d %f\n",y/x, ((float)y)/x);
/* affiche 1 1.333333 */
```

Priorité des opérateurs

Arité	Associativité	Par priorité décroissante
_ 2	GD	() [] -> .
1	DG	! ++ -- + - (type) * & sizeof
2	GD	* / %
2	GD	+ -
2	GD	< <= > >=
2	GD	== !=
2	GD	&&
2	GD	||
3	DG	?:
2	DG	= += -= *= /= %=
2	GD	,

Évaluation des opérations

- L'ordre d'évaluation des opérandes d'un opérateur n'est pas précisé

```
int x, y = 1;
x = (y * 2) + (++y);  /* ambigu */
```

- L'ordre d'évaluation des arguments d'une fonction n'est pas précisé

```
printf ("%d %d\n", x++, fact(x)); /* ambigu */
```

- L'ordre d'évaluation des opérateurs commutatifs et/ou associatifs n'est pas précisé : (x+y)+z peut être évalué x+(y+z), malgré le parenthésage.

Plan

45

- Généralités sur le C
- Programmes en C
- Variables et constantes
- Structures de contrôle
- Opérateurs
- **Exercices**

Exercices

46

- Écrire la fonction puissance déclarée par

```
int puissance (int x, unsigned char n)
```

qui calcule la puissance n de x

- L'intégrer à un programme qui affiche pour chacun des nombres 2, 4, 6 … 12 ses puissances 1 à 5 (une ligne par nombre)

```
2    4    8    16   32
4    16   64   256  1024
```

…

(Exemple1.3.c)

Exercices

47

- Les nombres de Fibonacci sont définis par $F_0 = 1$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ (n>=2)
- Afficher la liste des 40 premiers nombres de Fibonacci (un nombre par ligne)
  1. En définissant une fonction récursive permettant de calculer le nombre de Fibonacci de rang n `(Exemple1.4.c)`
  2. De façon itérative (sans définir de fonction auxiliaire) `(Exemple1.5.c)`

