Eléments d'Informatique

Cours 1- Représentation des nombres
Notion de variable

Catherine Recanati

UNIVERSITÉ PARIS 13 NORD

Plan général

- **Représentation des nombres. Notion de variable.**
- Programme. Expressions.
- Architecture des ordinateurs: langage machine, langage assembleur, AMIL.
- Systèmes d'exploitation : fichiers, processus, compilation.
- Instructions de contrôle: boucles et branchements.
- Programme, définition de fonction, appel fonctionnel.
- Tableaux de variables et fonctions d'arguments de type tableau.
- Sens d'un programme, pile d'exécution, compilation.
- Pointeurs et tableaux.
- Chaines de caractères, bibliothèque <string.h>.
- Allocation dynamique, liste chaînées.
- Révisions.

- Cours 1 -
Représentation des nombres
Notion de variable

- Codage binaire des entiers
- Codage hexadécimal
- Notion de variable
- Types de variables

Représentation des entiers

En base 10 (décimale), on utilise 10 chiffres: 0,1,2,...6,7,8 et 9.
Ainsi, si n = 17, on aura $Rep_{10}(n) = 17$, car

$$n = 1 \times 10^1 + 7 \times 10^0$$

En base 2 (binaire), on n'utilise que 2 chiffres: 0 et 1. La décomposition de n en somme de puissances de 2

$$n = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

fournit $Rep_2(n) = 1001$.

Codage binaire des entiers

Ainsi, en binaire:

1	s'écrit	1	$1x2^0$
2	s'écrit	10	$1x2^1+0x2^0$
3	s'écrit	11	$1x2^1+1x2^0$
4	s'écrit	100	$1x2^2+0x2^1+0x2^0$
5	s'écrit	101	$1x2^2+0x2^1+1x2^0$
6	s'écrit	110	$1x2^2+1x2^1+0x2^0$

etc.

Plan

Codage binaire des entiers

1	s'écrit	1	
2	s'écrit	10	
2^2 -1	s'écrit	11	
2^2	s'écrit	100	
2^3 -1	s'écrit	111	
2^3	s'écrit	1000	
…	…	…	
2^n -1	s'écrit	111… …11	(n chiffres 1)
2^n	s'écrit	100… …00	(un 1 + n zéros)

Décimal ➔ Binaire

Pour trouver la représentation binaire d'un nombre, il faut le décomposer en somme de puissances de 2.

Il est donc intéressant de connaitre un peu la liste des puissances de 2:

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024,

…, 16 384, …, 65 536, etc.

Plan

L'addition en binaire

```
 1 + 1 = 10
10 + 1 = 11
11 + 1 = 100
```

```
  1011
+   11
  -----
  1110
```

« 1 + 1 = 10, je pose 0 et je retiens 1.
1+ 1 +1 = 11, je pose 1 et je retiens 1.
0 + 1= 1, je pose 1 et j'abaisse le 1 »

Codage hexadécimal

On utilise 16 caractères : les dix premiers chiffres 0,1,2,...6,7,8,9 et les six premières lettres : A, B, C, D, E, F. Ainsi A désigne 10, et F désigne 15.

ex: FFFF désigne le nombre 65 535.

Si n = 17, la décomposition de n en somme de puissances de 16

$$n = 1 \times 16^1 + 1 \times 16^0$$

nous donne sa représentation hexadécimale

$$Rep_{16}(n) = 11$$

Codage hexadécimal

Inversement, pour convertir en décimal une représentation hexadécimale, on calculera la somme des puissances de 16 correspondant aux différents coefficients hexadécimaux.

Ex: $Rep_{16}(n) =$ AF2B.

$n = A \times 16^3 + F \times 16^2 + 2 \times 16^1 + B \times 16^0$

$= 10 \times 4096 + 15 \times 256 + 2 \times 16 + 11 \times 1$

$= 40\,960 + 3\,840 + 32 + 11$

$= 44\,843$

Binaire ➔ Hexadécimal

Pour convertir une représentation binaire en hexadécimale, il suffit de considérer les digits (0 ou 1) par paquet de 4, et de les convertir en digits hexadécimaux :

Ex: $Rep_2(n) =$ `1111001011100100 10`

$Rep_2(n) =$ `11 1100 1011 1001 0010`

`3 12 11 9 2`

`3 C B 9 2`

$Rep_{16}(n) =$ `3CB92`

Hexadécimal ➔ Binaire

On transformera chaque digit hexadécimal par son code binaire (sur 4 bits) pour obtenir la représentation binaire d'un nombre à partir de sa représentation hexadécimale.

Ex: $Rep_{16}(n) =$ `FF2A`

$Rep_{16}(n) =$ `F F 2 A`

$Rep_2(n) =$ `1111 1111 0010 1010`

car $Rep_2(F) =$ `1111`, $Rep_2(2) =$ `0010` et $Rep_2(A) =$ `1010`

Plan

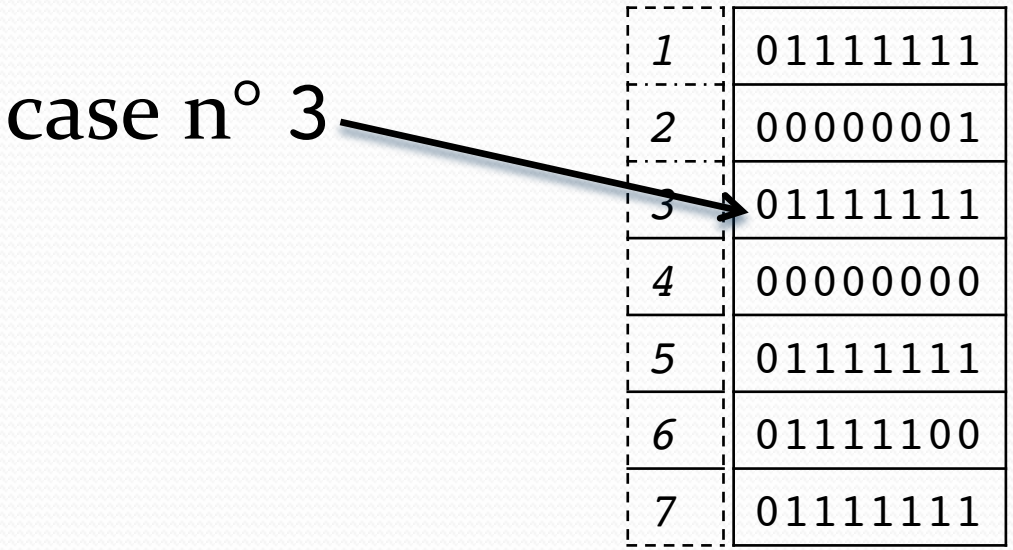
Mémoire

La mémoire d'un ordinateur est constituée d'unités élémentaires, les bits. Un *bit* (de **b**inary un**it**) vaut `0` ou `1`. Un ensemble de 8 bits consécutifs s'appelle un *octet*.

Le *byte* est la plus petite unité de mémoire adressable (souvent 8 bits).

Notion de variable

Une variable est une « case » ou zone mémoire dont on connait l'adresse de début et l'étendue (sa taille en nb de bytes).



Notion de variable

Une variable possède un nom, son identificateur, et une valeur (encodée par une suite de bits).

`angle`

0	0	0	0	0	1	0	1

Avec 8 bits, on pourra coder les entiers positifs compris entre `0` et 2^8-1, c'est-à-dire `255`.

Ici `angle` a pour valeur entière `5`.

Déclaration de variable

On déclare une variable en déclarant son **identificateur** (= son nom), et son **type** (= celui de ses valeurs encodées sur un certain nb de bits).

Un *identificateur* est défini syntaxiquement par la notation (BNF):

ident ::= lettre (lettre | chiffre)*

lettre ::= **a** | **b** |…|**z** | **A**| …|**Z** |**_**

chiffre ::= **0** | **1** | **2** | … | **9**

exemples: x, _BiDoN, jour_ferie

Types de variables

On dispose en C de plusieurs types d'entiers :

`int`	entiers signés
`unsigned int`	entiers positifs
`long int`	entiers longs
`unsigned long int`	entiers longs positifs
`short int`	entiers petits
`unsigned short int`	entiers positifs petits

La taille d'un `int` varie selon les ordinateurs, mais on aura souvent 8 bits pour les `short`, 16 (ou 32) bits pour les `int`, et 32 (ou 64) bits pour les `long`.

Si les `short int` sont stockés sur 8 bits, ils permettent de coder les entiers relatifs appartenant à [-127, 127] ; et les `unsigned short int` permettent de coder les entiers positifs de 0 à 255.

ATTENTION aux débordements en cas d'addition : les bits supplémentaires seront ignorés et perdus. Le programme sera peut-être faux !

Types de variables

On peut aussi coder en C des nombres réels (en fait rationnels).

Il existe trois types de flottants, de la plus faible à la plus grande précision: `float, double et long double.`

La précision effectivement utilisée pour chacun de ces types dépend de l'implémentation.

Type char

Le type `char` désigne les caractères. Le plus souvent ils sont encodés par le code américain standard `ASCII` (sur 8 bits), et sinon `unicode` (sur 16 bits).

Les constantes de type `char` sont notées entre apostrophes (*quote*):

‘a’ désigne le caractère a.

‘`\012`’ désigne le caractère de code octal 12, c’est-à-dire 10.

Code ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[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	[NEGATIVE 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	[ENG OF 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	[SUBSTITUTE]	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	|

Déclarations de variables en C

```
    /* quelques variables numériques */
short int angle_mort ;
unsigned long grandeDistance ;
float taille;

    /* un caractère */
char lettre ;

    /* définitions de constantes */
#define  EXIT_SUCCESS     0
#define  BEEP             '\007'
```

Déclarations de variables en C

Pour les identificateurs (= nom des variables), on suit la syntaxe décrite précédemment.

Mais certains mots ne peuvent pas être utilisés. Ce sont les mots réservés (ou mots clés) du langage C. Nous avons déjà vu les noms des types de base, mais il y a d'autres mots réservés, que nous introduirons au fur et à mesure.

Mots réservés du langage C

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

Pour conclure : une variable possède

- une **adresse**. Sa localisation en mémoire lors d'une exécution du programme.
- un **type**. C'est celui des valeurs que la variable peut prendre (`int`, `char`, etc.). Ce type a une taille (sizeof (type) qui définira le nombre de bits (différent selon les plateformes) sur lesquels sera encodée la valeur de la variable. C'est un type d'allocation mémoire.
- une **valeur**. Elle est encodée par les bits présents à l'adresse de la variable.
- un **identificateur**. C'est le nom de la variable dans le texte du programme.

Plan

Merci pour votre attention.

Des questions ?

Les opérateurs utilisés pour décrire les constructions dans le format BNF sont :

– (a) : les parenthèses permettent de délimiter une expression BNF à laquelle un opérateur peut s'appliquer.

– a* : l'étoile représente la répétition (d'une construction a) un certain nombre de fois, éventuellement zéro.

– a+ : le symbole + représente la répétition(d'une construction a) un certain nombre de fois, dont au moins une.

– a | b : représente soit une construction décrite par a, soit une construction décrite par b (c'est soit a, soit b).

– [a] : les crochets indiquent que la présence de la construction a est optionnelle (a apparaît 0 ou 1 fois).

où a , b sont des descriptions BNF de constructions possibles

Les symboles terminaux sont représentés dans une typographie particulière. On a utilisé ici des caractères gras et de couleur rouge, (comme **2**) dans les diapos précédentes.