Séance 2

Bases de la programmation, partie 2

Sébastien Combéfis

mercredi 17 septembre 2014

Rappels du cours précédent

- Programme, chaine de compilation
- Exemple *« Hello World »*
- Fonction principale, **#include**, printf , commentaire
- Variable : déclaration, initialisation, type, valeur
- Opérations de base : $+$, $-$, $*$, $/$, $\%$
- Conditions : $>$, $<$, $>=$, $<=$, $==$, $!=$
- Instructions **if** et **if**$-$**else**
- Boucles **while** et **for**

Types de données

- Nombre entier : **short**, **int**, **long int**, **long long**

  Nombre d'habitants d'un pays, nombre d'élèves d'une classe

- Nombre flottant : **float**, **double**, **long double**

  Un prix, la consommation d'une voiture en litre, ...

- Caractère : **char**

  Une lettre, le sexe d'une personne (H/F)

Le type int

- Un **int** représente un nombre entier
- **signed int** permet des valeurs positives et négatives

  Valeur par défaut
- **unsigned int** permet de limiter aux valeurs positives

```
1  signed int a;
2  a = -10;
3
4  unsigned int b = 20;
5  a = a + b;
6
7  int c = -5;                    // Même chose que signed int
```

Représentation binaire des entiers

- Un nombre binaire est une somme de puissances de 2
- Rappel pour un nombre décimal, par exemple 1053

10^3	10^2	10^1	10^0
1000	100	10	1
$1\times$	$0\times$	$5\times$	$3\times$

$\Rightarrow 1000 + 50 + 3 = \mathbf{1053}$

- Par exemple, pour le nombre binaire 10011

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
$1\times$	$0\times$	$0\times$	$1\times$	$1\times$

$\Rightarrow 16 + 2 + 1 = \mathbf{19}$

Bit de signe

- On utilise le bit de poids fort comme bit de signe

  0 *pour un positif et* 1 *pour un négatif*

- Les autres bits sont utilisés pour représenter le nombre binaire

- Par exemple, 42 en binaire s'écrit 101010

  Sur 8 *bits,* 42 *s'écrit donc* $\mathbf{0}0101010$
  et -42 *s'écrit donc* $\mathbf{1}0101010$

- Sur n bits, on peut représenter $2 \cdot 2^{n-1} - 1$ nombres différents

  Allant de $-(2^{n-1} - 1)$ *à* $(2^{n-1}) - 1$

Complément à deux

- On obtient le complément à un en inversant tous les bits
- On ajoute ensuite 1 au résultat, en ignorant les dépassements
- Par exemple, sur 8 bits, on obtient la représentation de -42 comme suit :

  Sur 8 *bits,* 42 *s'écrit donc* 00101010
  Le complément à un est 11010101
  Et on ajoute 1 *pour avoir le complément à deux* 11010110

- Sur n bits, on peut représenter $2 \cdot 2^{n-1}$ nombres différents

  Allant de $-(2^{n-1})$ *à* $(2^{n-1}) - 1$

Propriétés

- Addition et soustraction de nombres en complément à deux
- Dépassement de capacité pour l'addition

  Deux opérandes du même signe et résultat du signe opposé
  Deux opérandes de signe opposé, jamais de dépassement

- Extension de signe pour représenter un même nombre sur plus de bits

  On répète le bit de signe

Occupation mémoire d'un type de donnée

- La fonction **sizeof**() donne l'espace mémoire d'un type
- L'espace mémoire occupé est exprimée en octets

```
1 printf ("Un int occupe %ld octets\n", sizeof (int));
2
3 printf ("Un char occupe %ld octets\n", sizeof (char));
```

Le type `float`

- Un **float** représente un nombre flottant (nombre à virgule)
- On ne sait pas représenter tous les nombres réels
- Les calculs sont parfois approximatifs

```
1  float f = 0.123;
2  f = f + 2.001;
3
4  float e;
5  e = 5;
6  e = 4.0;
```

Le type char

- Un **char** représente un caractère
- Un caractère n'est rien d'autre qui nombre entier

  La correspondance nombre $\leftrightarrow$ caractère
  est faite par une table de caractères

- Lors de l'initialisation, ne pas oublier les ' '

```
1 char c = 'A';
2
3 char e;
4 e = 65; // A vaut 65 dans la table ASCII
```

La table ASCII

- Table ASCII (iso-646) 7 bits

  Nombre entier entre 0 *et* 127

	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	spc	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Espace mémoire des types de données

- Nombre entier
  - **short** : 2 octets
  - **int** : 4 octets
  - **long int** : 8 octets
  - **long long** : 8 octets
- Nombre flottant
  - **float** : 4 octets
  - **double** : 8 octets
  - **long double** : 16 octets
- Caractère
  - **char** : 1 octet

Limites I

- limits.h permet d'avoir des informations sur les types de données

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

int main()
{
    // Affiche la valeur max d'un int
    printf ("%d\n", INT_MAX);

    // Affiche la valeur min d'un int
    printf ("%d\n", INT_MIN);

    return 0;
}
```

Limites II

- Nombre entiers

Type	Non signé	Signé
short	0 à 65 535	−32 768 à 32 767
int	0 à 4 294 967 295	−2 147 483 648 à 2 147 483 647
long int	0 à 18 446 744 073 709 551 616	−9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

- Nombres flottants

Type	Valeurs possibles
float	$3{,}4 \cdot 10^{-38}$ à $3{,}4 \cdot 10^{38}$
double	$1{,}7 \cdot 10^{-308}$ à $1{,}7 \cdot 10^{308}$

Approfondissement de `printf`

- On peut limiter le nombre de chiffres après la virgule pour les nombres flottants

  `%.5f` *affichera 5 décimales*

- On peut ajouter des espaces ou des 0 devant des nombres

  `%05d` *affichera au moins 5 chiffres en ajoutant des 0 devant si nécessaire*

```
1  float f = 1.2345678;
2  printf ("%.3f", f);      // Affiche "1.234"
3
4  int a = 4;
5  printf ("%3d", a);       // Affiche "  4"
6  printf ("%03d", a);      // Affiche "004"
```

L'opérateur modulo

- a % N : son résultat se situe dans l'intervalle [0 ; N[
- On peut tester la parité d'un nombre avec % 2

  Le résultat sera 0 si le nombre est pair, sinon 1

- On peut tester la divisibilité par *N* d'un nombre avec % N

```
1  int a = 4 % 2;     // a vaut 0 car 4 est pair
2
3  int b = 3 % 2;     // b vaut 1 car 3 est impair
4
5  int c = 20 % 4;    // c vaut 0 car 20 est divisible par 4
```

Nombre pseudo-aléatoire

- Initialisé avec une graine avec srand (X)

  X est un nombre entier

- Après l'initialisation, on peut obtenir le nombre pseudo-aléatoire suivant avec rand()

```
1  #include <stdlib.h> // Obligatoire pour srand et rand
2
3  int main()
4  {
5      srand (3); // générateur initialisé avec la graine 3
6
7      int a = rand(); // a vaut un nombre aléatoire
8      int b = rand(); // b vaut un nombre aléatoire
9
10     return 0;
11 }
```

Exemple d'utilisation du modulo

- Il est possible de ramener un nombre entier dans l'intervalle [min ; max]

$$(X \ \% \ (max - min + 1)) + min$$

```
1  srand (10);
2
3  int a = (rand() % 4) + 2;      // Intervalle [2; 5]
4
5  int b = (rand() % 7) + 3;      // Intervalle [3; 9]
```

Opérateurs logiques

- Trois opérateurs logiques : && (ET), || (OU) et ! (NON)

a	b	! a	a && b	a || b
0	0	1	0	0
0	1		0	1
1	0	0	0	1
1	1		1	1

```
1 int a = 2 == 2 && 3 == 3;     // a vaut 1
2 int b = 2 == 3 || 3 == 3;     // b vaut 1
3 int c = !(2 == 3) && 3 == 3; // c vaut 1
```

Simplification de conditions

- ! (x != b) est équivalent à (x == b)
- ! (x > a) est équivalent à (x <= a)
- ! (x >= a) est équivalent à (x < a)
- Règles de De Morgan
  - !(a && b) est équivalent à (!a || !b)
  - !(a || b) est équivalent à (!a && !b)

Opérations arithmétiques raccourcies

- Incrémentation et décrementation
  - a = a + 1 est équivalent à a++
  - a = a − 1 est équivalent à a−−
- Opérateurs arithmétiques
  - a = a + b est équivalent à a += b
  - a = a − b est équivalent à a −= b
  - a = a ∗ b est équivalent à a ∗= b
  - a = a / b est équivalent à a /= b
  - a = a % b est équivalent à a %= b

Opérateurs binaires

- ET binaire (&), OU binaire (|) et NON binaire (~)
- OU exclusif (^)
- Décalage de bits à gauche (<<) ou à droite (>>)

Afficher les bits d'un nombre entier

- On décale 1 de i bits vers la gauche
- On fait le ET binaire pour ne garder que le bit à la i^{e} position
- On décale le résultat de i bits vers la droite pour avoir la valeur du i^{e} bit

```
1  unsigned int value = 42;
2
3  int i;
4  for (i = 8 * sizeof (unsigned int) - 1; i >= 0; i--)
5  {
6      printf ("%d", (value & (1 << i)) >> i);
7  }
```