Introduction au langage C - Cours 1

Girardot/Roelens

Septembre 2012

Le contenu

Première partie du cours Langages et Concepts de Programmation

- concepts élémentaires de la programmation
- une introduction au langage C
- une introduction à l'algorithmique
- une introduction à l'utilisation des ordinateurs

La forme

- un élève par poste (demi-promo)
- période « bloquée » intensive
- 7 séances : 1h30 de cours, 1h30 de TP
- 1 mini-projet : 4h30 le dernier jour

Les documents

- polycopié (contenant les sujets des TP)
- copie des transparents (prise de notes)
- **corrigé d'exercices et de projets antérieurs**
- site web : `http://kiwi.emse.fr/INTROINFO/`

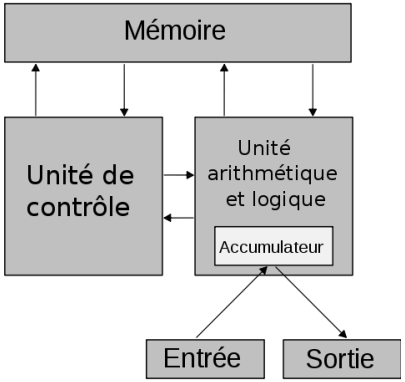
Les intervenants

- Marc Roelens, responsable du cours
- enseignants-chercheurs et doctorants en informatique (Institut Henri Fayol)

Contrôle des connaissances

- Évaluation du projet
- travail individuel

Structure de l'ordinateur



L'architecture de Von Neumann

Processeur

Processeur, ou unité centrale, ou CPU
(*Central Principal Unit*)

- Unité de commande
  - transfert des instructions depuis la mémoire
  - décodage et exécution des instructions
- Registres
  - mémoires très rapides situées dans l'UC
  - registres adresses : désignent un élément de la mémoire
  - registres données : représentent une valeur
  - registres d'état : représentent une partie de « l'état » du processeur

- Unité arithmétique et logique
  - décodage des fonctions
  - opère sur les registres
  - opérations arithmétiques de base : $+ - \times \div$
  - opérations logiques, décalages : $\vee \wedge \sim, \ll, \gg$
- Performances
  - liées à la fréquence d'horloge et à la nature du processeur
  - mesurées en **MIPS**, *Millions of Instructions Per Second*
  - ...ou en **MFLOPS**, *Millions of Floating point Operations Per Second*

Mémoire

Cases numérotées

- taille standard d'une case : 8 bits (octets)
- toute valeur écrite dans une case persiste tant qu'elle n'est pas modifiée (et que la machine fonctionne)
- on appelle *adresse* le numéro de la case

Contenu

- instructions du programme
- données du programme

Instructions du processeur

Langage machine : l'ensemble des instructions acceptées par un processeur donné

- Chaque processeur a son propre langage machine

Caractéristiques communes :

- instructions conservées en mémoire centrale
- composées d'un « code opération », de désignation de registres et/ou d'adresses en mémoire
- représentées par des nombres, tenant sur 1, 2, 4... 8 octets
- exécutées en séquence, sauf instructions « de saut »
- transferts entre les registres et la mémoire centrale
- opérations arithmétiques et logiques sur les registres
- tests et positionnement des registres d'état
- commandes des périphériques
- instructions de « saut », conditionnel ou non

Le langage d'assemblage

Une représentation « lisible » des instructions de la machine

- les codes opérations et les registres sont représentés par des mnémoniques
- l'assembleur permet d'associer des « étiquettes » symboliques à des adresses en mémoire
- un logiciel spécialisé (« assembleur ») traduit les instructions d'assemblage en langage machine.

Exemple (assembleur 68000) :

```
            move.l    4(a7),d0
            sub.l     8(a7),d0
            bvc.s     suite
            or.l      #1,ovfl(a5)
suite :     rts
```

Format typique : étiquette optionnelle, code instruction, opérandes éventuels

Représentation des données

Objets manipulés par les programmes

- objets « élémentaires » : nombres, caractères
  - nombres entiers, représentés par 1, 2, 4, ou 8 octets
  - nombres flottants, représentés par 4, 8 ou 16 octets
  - caractères :
    - code ASCII (128 caractères, 1 caractère = 1 octet)
    - code(s) ISO (256 caractères, 1 caractère = 1 octet)
    - code UNICODE (tous les caractères du monde, 1 caractère = 2 ou 4 octets)
- objets « complexes » : convention d'assemblage des objets élémentaires

Le code ASCII

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Lecture : A a pour code binaire 100 0001, soit en décimal 65

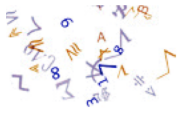
Mémoire secondaire

Mémoire vive :

- les données disparaissent lorsque le courant est coupé
- ne peut stocker la totalité des données utilisées par un ordinateur

Mémoire secondaire :

- utilise des supports pérennes (plusieurs années) ;
- taille d'un ordre de magnitude supérieure à la taille de la mémoire vive



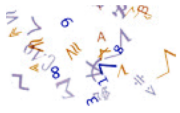
Fichiers

Systèmes de fichiers

- plaquer une organisation « logique » sur l'espace physique disponible
- permettre un adressage symbolique (par nom) à des groupes de données : les *fichiers*
- fournir un index des fichiers disponibles
- gérer des informations supplémentaires (date de dernière modification, possesseur, droits d'accès...)

- organiser les fichiers sous forme hiérarchique (arborescence) qui en permet la classification
- fichiers spécialisés : les répertoires (descriptions de fichiers et de répertoires)
- autres fichiers « banalisés » : pour le système, un fichier est un (gros) tas d'octets
- les fichiers sont souvent désignés par un nom comportant un *suffixe spécifique*, dit « extension », qui permet d'en deviner la nature : `toto.c`, `rapport.doc`...

```
c:\durant\coursc\toto.c
/users/durant/coursc/toto.c
```

Utilisation des ordinateurs

L'ordinateur nu est totalement inexploitable

Un ordinateur dispose d'un mécanisme d'initialisation, qui permet de charger un programme particulier, le système d'exploitation

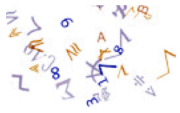
Le système d'exploitation gère pour l'utilisateur les ressources de la machine et en assure une vision « conviviale »

Il existe de multiples systèmes d'exploitation, aux finalités assez voisines

- Windows (différentes versions)
- UNIX (porté sur la totalité des machines)
- Linux (un UNIX écrit spécifiquement pour les PC)
- Mac-OS (spécifique aux Macintosh)

Compilation

- le langage machine est complexe, difficile à maîtriser, dépendant de la machine
- on construit les programmes dans des langages plus réguliers, de haut niveau, que l'on transforme en langage machine
- **compilation** : le processus qui transforme un programme écrit dans un langage de haut niveau en un ensemble d'instructions exécutables par la machine
- **compilateur** : le programme qui effectue le processus de compilation
  - ex : `gcc` pour le langage C
  - les objets manipulés par le compilateur sont des fichiers :
    - texte source, représenté en ASCII, « lisible »
    - texte exécutable, représenté dans un format propre au système d'exploitation

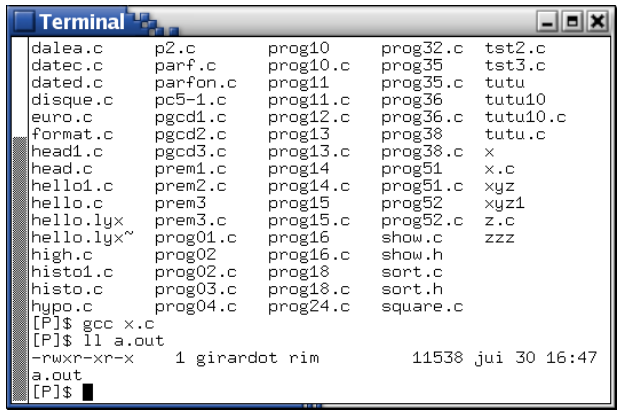


Création d'un programme

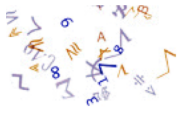
Le processus de création d'un programme consiste à :

- se placer dans une fenêtre de commandes
  - Linux : Terminaux⇒XTerm
- créer un fichier contenant le texte du programme source en langage de haut niveau, au moyen d'un **éditeur de texte**
  - Linux : `Gedit`, `Kwrite`, voire `vi`, `vim`, `emacs`
- compiler le programme au moyen du **compilateur** adéquat
  - Linux : `gcc` pour le langage C
- lancer l'exécution du **programme exécutable**

Fenêtre de commande



XTerm Linux



Fenêtre de commandes

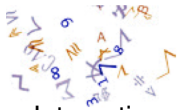
- permet des interactions en mode ligne
  - la commande est lancée par la saisie au clavier d'une ligne de texte terminée par la touche Entrée
  - les informations fournies à la commande sont saisies sous forme de lignes de texte terminées par la touche Entrée
  - les informations fournies par la commande sont affichées dans le terminal sous forme de lignes de texte terminées par un saut de ligne
- pas d'utilisation de la souris
- permet d'écrire facilement des petits programmes de commandes (*scripts*)

Fenêtre de commandes (2)

Quelques commandes :

exit	interrompre l'interaction
pwd	position du répertoire courant
ls	liste des fichiers du répertoire courant
.	représente le répertoire courant
..	est la racine du répertoire (« monter » d'un niveau)
/	est la racine du système (niveau le plus haut)
cd	changement de répertoire

```
[tp] cd ../durand/cours
[tp] cd  [tp] ls  foo.c toto.c
```



Fenêtre de commandes (3)

Interactions en mode ligne

cp nom1 nom2 recopier un fichier

```
[tp] cp prog2.c ../durand
```

rm nom supprimer un fichier

mv nom1 nom2 renommer ou déplacer « nom1 »

mkdir nom créer un nouveau répertoire

rmdir nom supprimer un répertoire

```
[tp] mv toto.c truc.c
```

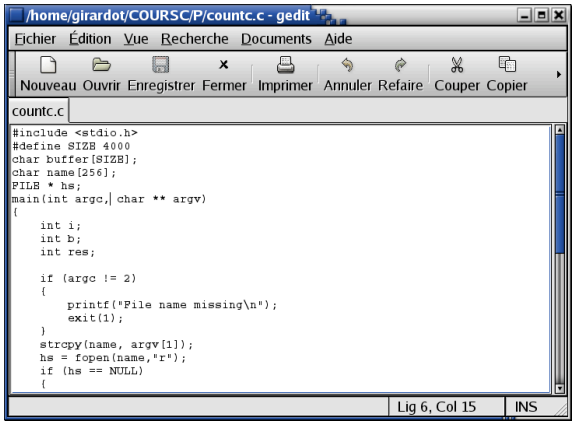
toto exécuter le programme `toto`

```
[tp] toto
[tp] gcc -o titi titi.c
```

./toto exécuter le programme `toto`

```
[tp] ./toto
```


Fenêtre d'édition



Fenêtre GEdit

Un premier exemple

Le programme *Hello world*

```
/* Premier programme */
#include <stdio.h>
int main(int argc, char * argv [])
{
    printf("hello world\n");
    return 0;
}
```

Son exécution

```
[P]$ gcc -Wall -ansi -pedantic prog1.c -o prog1
[P]$ ./prog1
hello world
[P]$
```


Éléments de C (1)

Format d'un programme

- format des instructions : libre
- blancs et tabulations utilisés pour rendre lisible le programme
- commentaires : placés entre `/*` et `*/`
- cas particuliers
  - les « commandes » pour le pré-processeur (débutant par `#`) doivent être sur une ligne unique
  - le programme doit se terminer par un saut de ligne (oublié par certains éditeurs de texte)

Éléments de C (2)

Chaîne de caractères : `"hello world\n"`

- c'est une donnée du langage (constante)
- représentée par une suite de caractères, placée entre « double quotes », "
- le caractère « back-slash » `\` permet de représenter des caractères particuliers :
  - `\n` : un passage à la ligne
  - `\t` : une tabulation horizontale
  - `\"` : une double quote
  - `\\` : un « back-slash »

Éléments de C (3)

Impression de résultats

- procédure du système `printf`
- nécessite l'inclusion de `stdio.h`
- premier élément : un *format* sous forme d'une chaîne de caractères
- autres éléments : les valeurs à imprimer
- fonctionnement :
  - les caractères du format sont recopiés en sortie
  - le caractère `%` introduit l'impression d'une valeur
  - le type de la valeur est défini par le caractère qui suit le `%`
    - **`d`** : un nombre « décimal » (un `int`)
    - **`f`** : un nombre « flottant » (un `float`)
  - la séquence **`\n`** provoque un passage à la ligne

Exemple

```
int A=3, B=17;
printf("A = %d\nB =  %d\n",A,B);
```

imprime

```
A = 3
B = 17
```


Un second exemple

Calcul de la surface d'un disque de diamètre donné

$$S = \pi \frac{D^2}{4}$$

Programme

```
/* Surface d'un disque */
#include <stdio.h>
int main(int argc, char * argv [])
{
float diametre=12.25,surface;
float pi=3.1415926535;
surface=pi*(diametre/2)*(diametre/2);
printf("La surf. du disque de diam. %f est %f\n",
        diametre,surface);
return 0;
}
```


Variables

- représentent les données du programme
- désignées par un nom dont le premier caractère est alphabétique : {`A-Z,a-z,_`}, les suivants sont alphanumériques : {`A-Z,a-z,0-9,_`}
- ont un *type* : pour l'instant, `int` ou `float`

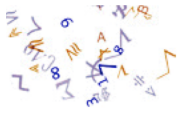
Valeurs numériques ou « constantes »

- nombres « entiers » ou « flottants »
- entiers : `0 72451 -200`
- flottants : `0.0 127.0 -0.246 -1.25e8 1e-17`

Déclarations

- toute variable doit être déclarée
- une déclaration consiste en un type suivi d'une ou plusieurs variables
- à une variable peut être associée une valeur initiale
- exemple :

```
int toto, titi=245, tutu;
float facteur=6.55957;
```



Quelques types en C

Mots clefs	Description	Taille
`int`	entier	4 octets
`long`	entier	4 octets
`short`	entier	2 octets
`char`	entier	1 octet
`float`	flottant	4 octets
`double`	flottant	8 octets

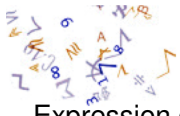
$$sizeof(char) \leq sizeof(short) \leq sizeof(int) \leq sizeof(long)$$

$$sizeof(float) \leq sizeof(double)$$

Expressions

Une expression permet de calculer une valeur dans le programme

- constante (valeur = valeur de la constante)
- nom d'une variable (valeur = contenu à cet instant de la variable)
- expression arithmétique
  - opérandes : expressions
  - opérateurs : `+ - * / %`
    - priorité des opérateurs :
      moins unaire ≫ multiplication, division et modulo ≫ addition et soustraction
    - ordre d'évaluation de gauche à droite pour une même priorité
  - parenthèses : modifient l'ordre des opérations
  - type homogène : entier *ou* flottant



Expressions (2)

Expression d'affectation

- syntaxe : « variable » `=` « expression » `;`
- types de la variable et de l'expression homogènes (ou conversion implicite)
- valeur : valeur de l'expression
- effet : enregistre la valeur de l'expression dans la variable
- ordre d'évaluation : de droite à gauche

  ```
  e = a = (b=3)+(c=(d=4)+2)
  ```

  `d` prend la valeur 4, `c` prend la valeur 6, `b` prend la valeur 3, `a` prend la valeur 9, `e` prend la valeur 9, la valeur globale de l'expression est 9

Une expression suivie d'un point-virgule est une **instruction** du programme

```
24+a;
a=3;
```

La première instruction est syntaxiquement correcte, mais ne fait rien (avertissement du compilateur)

Synthèse : éléments de C

- les inclusions du préprocesseur

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

- les déclarations de variables

  ```
  int toto; int titi=530;
  float facteur=6.55957;
  ```

- la procédure `main`

  ```
  int main(int argc, char * argv [])
  {
          « instructions »
  }
  ```

Expressions et instructions

Expressions :

- valeurs de variables
- expressions arithmétiques
- expressions d'affectation

Instructions :

- affectation : « variable » = « expression »`;`
- impression :

Développement en C

Écriture des programmes

- travailler dans un répertoire spécifique
- utiliser un éditeur de texte (gedit, kedit, vi, emacs...)
- respecter les conventions (`toto.c` `toto.h`...)

Compilation

```
$ gcc -ansi -pedantic -Wall -o toto toto.c
```

Exécution, mise au point

```
$ toto
$ ./toto
```

- interruption d'un programme par `Control-c`