Programmation

But du cours

- Comprendre les fondements des langages de programmation
  (portée des variables, appel par valeur/par référence, etc.)
- du cambouis…
  (binaire, assembleur, ASCII, etc.)
- à la sémantique
  (mathématiques, preuves de programmes)

Types de langages de programmation

- Impératifs: C, C++, autres?
- Fonctionnels: Caml, autres?
- Objets: Java, Python, autres?
- Logiques: Prolog, autre?

Factorielle en C, style impératif

```
int fact (int n)
{
   int resultat;
   int i;

   resultat = 1;
   for (i=1; i<=n; i++)
     resultat = resultat * i;
   return resultat;
}
```

Factorielle en Caml, style fonctionnel

```
let rec fact n =
        if n=0
           then 1
           else n * fact (n-1);;
```

En maths:

$$n! = \begin{cases} 0 & \text{si } n = 0 \\ n.(n-1)! & \text{si } n \neq 0 \end{cases}$$

Factorielle en Prolog

- On spécifie une *relation* entre entrées et sorties:

```
fact(0,1).
fact(N+1,Y) :- fact(N,Z), Y=(N+1)*Z.
```

Sémantique

- But 1: *savoir* ce que fait un programme
- But 2: *raisonner* mathématiquement sur les programmes (fait-il ce qu'on veut?)

Sémantique, question 1

- En C, que fait `x=x++;` ?

- Indication: `x++` prend `x`, l'incrémente, et retourne l'ancienne valeur de `x`.

Sémantique, question 2

```
int fact (int n)
{
  int resultat;
  int i;

  resultat = 1;
  for (i=1; i<=n; i++)
    resultat = resultat * i;
  return resultat;
}
```

```
let rec fact n =
        if n=0
           then 1
        else n * fact (n-1);;
```

- `fact` en C fait-il la même chose que `fact` en Caml?
- … pour quelle définition de « faire la même chose »?

Où est le bug?

- Un programme de tri fusion:

```
void merge (int *l1, int n1,
            int *l2, int n2,
            int *res) {
  while (n1!=0 && n2!=0) {
    if (*l1 < *l2)
       { *res++ = *l1++; n1--; }
    else { *res++ = *l2++; n2--; }
  }
  while (n1--!=0) *res++ = *l1++;
  while (n2--!=0) *res++ = *l2++;
}
```

```
void sort1 (int *l, int n,
            int *res) {
  int k;
  if (n==0) return;
  if (n==1) { *res = *l; return; }
  k = n / 2;
  sort1 (l, k, res);
  sort1 (l+k, n-k, res+k);
  merge (res, k, res+k, n-k, l);
}
```

```
void sort (int *l, int n) {
  int *aux = (int *) malloc (n * sizeof (int));
  sort1 (l, n, aux);
  free (aux);
}
```

Cambouis 1 : cat

- En Unix, les commandes sont… des programmes
- Par exemple, `cat` concatène les fichiers données en argument
- Lançons:
  `$ cat a b`

cat.c

```
#include <stdio.h>

main (int argc, char *argv[])
{
  int i, c;

  for (i=1; i<argc; i++)
    {
      FILE *f;

      f = fopen (argv[i], "r");
      while ((c = fgetc (f))!=EOF)
        fputc (c, stdout);
      fclose (f);
    }
  fflush (stdout);
  exit (0);
}
```

cat cat

- Tout programme est un fichier
- Affichons cat

```
$ cat `which cat`
```

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@
'\211^D^H4^@^@^@x!^@^@^@^@^@^@4^@ ^@^F^@(^@^X
^@^W^@^F^@^@^@4^@^@^@4\200^D^H4\200^D^HÀ^@^@^@
À^@^@^@^E^@^@^@^D^@^@^@^C^@^@^@ô^@^@^@ô\200^D
^Hô\200^D^H^S^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A
^@^@^@^@^@^@^@^@\200^D^H^@\200^D^H3^_^@^@3^_^@
^@^E^@^@^@^@^P^@^@^A^@^@^@4^_^@^@4¯^D^H4¯^D^Ht
^A^@^@,^B^@^@^F^@^@^@^@^P^@^@^B^@^@^@^H ^@^@^H
°^D^H^H°^D^H ^@^@^@ ^@^@^@^F^@^@^@^D^@^@^@^D^@
^@^@^H^A^@^@^H\201^D^H^H\201^D^H ^@^@^@ ^@^@^@
^D^@^@^@^D^@^@^@/lib/ld-linux.so.2^@^@^D^@^@^@
^P^@^@^@^A^@^@^@GNU^@^@^@^@^@^B^@^@^@^@^@^@^@
```

- compréhensible… pour le processeur

Démo

- On compile:
  `$ gcc -o mycat main.c`
- Ceci traduit du C
  (lisible par un ~~humain~~ geek)
  vers le langage machine (ou assembleur)
- Démo: exécution pas à pas

Langages impératifs

- **Affectations:** `x=e;`
  agit par *modification* de la mémoire
  attention: le test d'égalité en C, c'est ==!
- **Séquence:** cmd1; cmd2; …; cmd;
- **Tests:**
  `if` (cond) then-branch; [`else` else-branch;]
- **Boucles:** `while` (cond) body;

Langages impératifs

- Appels de fonctions
- Tableaux
- Pointeurs
- etc.

Types de base

- `int`: entiers machine signés ($-2^{63}..+2^{63}-1$)
- `unsigned int`: — non signés ($0..2^{64}-1$)
- `char`: octets (0..255)
- `float`, `double`: nombres ~~réels~~ flottants (i.e., à virgule flottante)
- … **tout** est suite de bits

Les entiers non signés

- Représentés sur 64 (ou 32, ou …) bits

```
391
== 0b... 0001 1000 0111
== 0x0000 0000 0000 0187
```

- Addition, multiplication, etc. interprétées modulo 2^{64}.
- D'où *débordement arithmétique*

Les entiers signés

- *En complément à 1:*
  on garde un bit de signe + 63 bits de valeur
  Problème(s)?
- *En complément à 2:*
  nombres interprétées mod 2^{64},
  comme avant,
  mais dans un intervalle décalé $-2^{63}..+2^{63}-1$
- Portabilité? …

Complément à 2

- Sur 4 bits, pour simplifier
- Les nombres sont –8…7:

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	1111	0	1	10	11	100	101	110	111
8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7

- Addition, multiplication: inchangées par rapport au cas non signé
- Miracle: que calcule `x & -x`? (si `x`≠0)

Les octets

- 8 bits (d'où le nom)
- type `char`, car sert à représenter les caractères
- Les notations suivantes désignent la même chose, en machine:

```
A   65   0x41   0b01000001
```

- Pourquoi?

Caractères: le code ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Inutilisé…

Caractères: le code ~~ASCII~~ iso-latin-1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80			‚	*ƒ*	„	…	†	‡	ˆ	‰	Š	‹	Œ			
90		‘	’	“	”	•	–	—	˜	™	š	›	œ			Ÿ
A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Unicode

- Années 1990: une foison de codes langues à besoins spécifiques: hongrois, islandais, arabe, hébreu, hindi, chinois, japonais, arménien, etc.

Klingon[1][2]

ConScript Unicode Registry code chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+F8Dx																
U+F8Ex																
U+F8Fx																

Notes

1. ^ As of 1997-02-14 version
2. ^ Grey areas indicate non-assigned code points

- Standard: Unicode (utf-8, utf-16, utf-32…)