Introduction au Langage C

Marie Pelleau
marie.pelleau@univ-cotedazur.fr

Basé sur les transparents de Jean-Charles Régin

Notes

Plan du cours

- 6 cours
- Contrôle des connaissances :
  - QCM : 20 %
  - Projet : 30 %
  - Contrôle terminal : 50%

Notes

Bibliographie

The C Programming Language

Kernighan B.W., Ritchie D.M., *Prentice Hall*, 1978

Le langage C - C ANSI

Kernighan B.W., Ritchie D.M., *Masson - Prentice Hall*, 1994, 2e édition
Traduit par J.-F. Groff et E. Mottier

Langage C - Manuel de référence

Harbison S.P., Steele Jr. G.L., *Masson*, 1990
Traduit en français par J.C. Franchitti

Notes

Langage C

- Inventé en 1972 par Dennis Ritchie
- Langage de bas niveau
- Créé pour porter des systèmes d'exploitation (Unix)
- But : un compilateur peut-être écrit en 2 mois
- A permis de réécrire des programmes assembleurs
- Programmation indépendante de la machine
- Portable : présent sur presque toutes les architectures ayant été inventées

Notes

Langage C

- Utilisé du micro-contrôleur au super-ordinateur
- Présent dans les systèmes embarqués
- Sont écrits en C
  - Unix
  - Linux
  - Windows
  - Tous les Compilateurs GNU
  - GNOME
  - Les machines virtuelles JAVA

- Un code C optimisé permet d'obtenir le code le plus rapide
- Fortran, C, C++ sont équivalents
- Java est un petit peu plus lent (mais pas beaucoup)

Notes

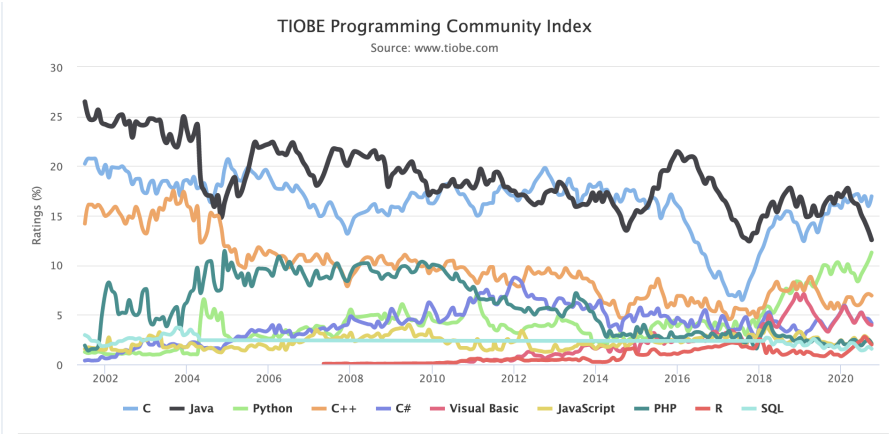
Tiobe : popularité des langages

Oct 2020	Oct 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.95%	+0.77%
2	1	↓	Java	12.56%	-4.32%
3	3		Python	11.28%	+2.19%
4	4		C++	6.94%	+0.71%
5	5		C#	4.16%	+0.30%
6	6		Visual Basic	3.97%	+0.23%
7	7		JavaScript	2.14%	+0.06%
8	9	↑	PHP	2.09%	+0.18%
9	15	⇈	R	1.99%	+0.73%
10	8	↓	SQL	1.57%	-0.37%
11	19	⇈	Perl	1.43%	+0.40%
12	11	↓	Groovy	1.23%	-0.16%
13	13		Ruby	1.16%	-0.16%
14	17	↑	Go	1.16%	+0.06%
15	20	⇈	MATLAB	1.12%	+0.19%
16	12	⇊	Swift	1.09%	-0.28%
17	14	↓	Assembly language	1.08%	-0.23%
18	10	⇊	Objective-C	0.86%	-0.64%
19	16	↓	Classic Visual Basic	0.77%	-0.46%
20	22	↑	PL/SQL	0.77%	-0.06%

TIOBE Programming Community Index
Source: www.tiobe.com

Notes

Tiobe : popularité des langages



Notes

Langage C

Très utilisé car

- Bibliothèque logicielle très fournie
- Nombre de concepts restreint
- Permet de gérer des exécutables qui n'ont besoin de rien pour s'exécuter et pour lesquels on peut contrôler parfaitement la mémoire utilisée (noyau de Système d'exploitation, logiciel embarqué)
- Contrôle de bas niveau : très puissant

Notes

Langage C

Avantages

- Nombreux types de données
- Riche ensemble d'opérateurs et de structures de contrôle
- Bibliothèque d'exécution standard
- Efficacité des programmes
- Transportabilité des programmes (plus facile si on respecte une norme)
- Liberté du programmeur
- Interface privilégiée avec Unix

Notes

Langage C

Inconvénients

- Pas d'objets
- Pas de gestion des exceptions
- Peu de contrôles (on peut tout faire : débordement de tableaux, ...)
- Faiblement typé (on peut toujours convertir)
- Peu devenir franchement complexe

Notes

Langage C

Langages inspirés du C

- C++
- Objective-C
- Java
- PhP

Notes

Langage C

Environnement de développement

- Allez jeter un œil sur la page wikipedia du langage C (en français et en anglais)
- Visual Studio community C++ est gratuit !
  - Définir des `.c` au lieu de `.cpp` pour avoir la compilation C
- Visual Code est gratuit

Notes

Langage C

- Langage normalisé (C99)
- L'apprentissage du C permet de mieux comprendre le fonctionnement
  - D'un ordinateur (CPU, mémoire, périphériques)
  - D'un OS
- Un informaticien se doit d'avoir fait du C dans sa vie (sera toujours utile)

Notes

Test

```
void f (int i) {
  i = 8;
}

int main (void) {
  int i = 5;
  f(i);
  k = i;
}
```

Valeur de `k` ?

Notes

But du cours

- Vous faire comprendre plein de choses sur la programmation et sur comment écrire un programme efficace

Notes

Le C et le “snobisme” en programmation

```
void strcpy (char* dest, char* src) {
  while (*dest++=*src++);
}
```

Notes

Langage C : pour débuter

- Un programme C est constitué d'un ou plusieurs fichiers sources suffixés par `.c` et `.h`, dans le(s)quel(s) une unique fonction `main` doit apparaître (ce sera le point d'entrée du programme)
- Seules des fonctions peuvent être définies
  - Pour définir une procédure, il faut déclarer une fonction renvoyant le type spécial `void`
- Pour renforcer le fait que la fonction n'ait pas de paramètres, mettre `void` entre les parenthèses

Notes

Quelques règles

Ce n'est pas obligatoire dans le langage mais suivez ces règles :

- On met toujours des {}
  ```
  if (x > 3) {y = 4;}
  ```
- On évite plusieurs instructions sur la même ligne
  ```
  i = i + 1; j = j + 2; /* on sépare sur 2 lignes */
  ```
- On évite plusieurs déclarations sur la même ligne
  ```
  int i, j = 2, k = 5; /* à éviter */
  ```


Notes

Quelques règles

Ce n'est pas obligatoire dans le langage mais suivez ces règles :

- Les variables sont écrites uniquement en minuscule
- Les macros ou constantes définies à l'aide de `#define` sont toutes en majuscules
- Les noms de fonctions commencent par une minuscule
- Les noms de fonctions utilisent l'une des 2 formes
  - Tout en minuscule avec des _ pour séparer `fahrenheit_to_celcius`
  - En " collant " tout et mettant des majuscules pour séparer les mots `fahrenheitToCelcius`

Notes

Un premier exemple

```c
int main (void) {
  int i;
  int x = 3;
  int y = 4; /* y doit être positif */
  double z = 1.0;
  i = 0;
  while (i < y) {
    z = z * x;
    i = i + 1;
  }
  return 0;
}
```


Notes

On compile et on exécute

Compilation

- Compilateur (`gcc`)
- Des options (`-Wall`)
- Fichier source en entrée (`monfichier.c`)
- Fichier en sortie (`a.out` sous Linux, `monfichier.exe` sous Windows)

```
gcc -Wall -pedantic -ansi monfichier.c
```

Ce programme C calcule x^y, `x` et `y` étant donnés (`x` vaut 3, et `y` vaut 4).
Il n'affiche cependant rien du tout

Notes

Affichage (écriture)

```
int x;
```

```
fprintf(stdout, "%d", x);
```

Écrit un entier dans le fichier stdout (sortie standard)

```
printf("%d", x);
```

Écrit directement sur la sortie standard

Notes

On affiche quelque chose

```c
#include <stdio.h>
int main (void) {
  int x = 3;
  int y = 4;
  double z = 1.0;
  fprintf(stdout, "x = %d, y = %d", x, y);
  while (y > 0) {
    z *= x; /* z = z * x; */
    y -= 1; /* y = y - 1; */
  }
  fprintf(stdout, "z = %.2f \n", z);
  return 0;
}
```


Notes

Compilation et exécution

- On compile et on exécute
  - `gcc -Wall -pedantic -ansi foo.c`
  - `./a.out`

    `x = 3, y = 4, z = 81.00`
- Le programme calcule 3^4 et affiche les valeurs de `x`, `y` et `z`
- En C, il n'y a pas d'instructions d'E/S
- En revanche, il existe des fonctions de bibliothèque, dont le fichier de déclarations s'appelle `stdio.h` (standard input-output, `.h` pour " headers ")
- Les fonctions de bibliothèque d'E/S font partie de la bibliothèque C : `libc`

Notes

Compilation et exécution

- Le compilateur C utilisé est celui du projet : `gcc`
- Du fichier source au fichier exécutable, différentes étapes sont effectuées :
  - le **préprocesseur** `cpp`
  - le **compilateur** C `cc` traduit le programme source en un programme équivalent en langage d'assemblage
  - l'assembleur a construit un **fichier** appelé **objet** contenant le code machine correspondant
  - l'**éditeur de liens** `ld` construit le programme exécutable à partir des fichiers objet et des bibliothèques (ensemble de fichiers objets prédéfinis)

Notes

Compilation et exécution

- Les fichiers objets sont suffixés par `.o` sous Unix et `.obj` sous Windows
- Les librairies sont suffixées par `.a` `.sl` `.sa` sous Unix et par `.lib` sous Windows
- Les librairies chargées dynamiquement (lors de l'exécution du programme et non pas lors de l'édition de liens) sont suffixées par `.so` sous Unix et `.dll` sous Windows

Notes

Options du compilateur

- `-c` : pour n'obtenir que le fichier objet (donc l'éditeur de liens n'est pas appelé)
- `-E` : pour voir le résultat du passage du préprocesseur
- `-g` : pour le débogueur symbolique (avec les noms des fonctions)
- `-o` : pour renommer la sortie
- `-Wall` : pour obtenir tous les avertissements
- `-lnom_de_bibliothèque` : pour inclure une bibliothèque précise
- `-ansi` : pour obtenir des avertissements à certaines extensions non ansi de `gcc`
- `-pedantic` : pour obtenir les avertissements requis par le C standard strictement ansi

Notes

Calcul d'une puissance

```c
#include <stdio.h>
double puissance (int a, int b) {
  /* Rôle : retourne a^b (ou 1.0 si b < 0) */
  double z = 1.0;
  while (b > 0) {
    z *= a; /* z = z * a */
    b -= 1; /* b = b - 1; */
  }
  return z;
}
int main (void) {
  fprintf(stdout, "3^4 = %.2f \n", puissance(3, 4));
  fprintf(stdout, "3^0 = %.2f \n", puissance(3, 0));
  return 0;
}
```


Notes

Définition de fonctions

- En C, on peut définir des fonctions
- La fonction principale `main` appelle la fonction `puissance`, afin de calculer 3^4 et affiche le résultat. Elle appelle aussi la fonction `puissance` avec les valeurs 3 et 0 et affiche le résultat

Remark

Pour compiler, là encore, il n'y a aucun changement

- `gcc -Wall -pedantic -ansi foo.c`
- `./a.out`
  `3^4 = 81.00`
  `3^0 = 1.00`

Notes

Déclarations : prototype

```
double puissance (int a, int b) {
  /* corps de la fonction puissance
     déclarations
     instructions */
}
```

- Si on utilise une fonction avant sa définition alors il faut la déclarer en utilisant ce que l'on appelle un prototype :
  `double puissance (int, int);`
- Le compilateur en a besoin pour s'y retrouver
- L'utilisation de prototypes permet une détection des erreurs sur le type et le nombre des paramètres lors de l'appel effectif de la fonction

Notes

Lecture au clavier

```
int x;
```

`fscanf(stdin, "%d", &x);`
Lit un entier dans le fichier stdin (entrée standard)

`scanf("%d", &x);`
Lit directement sur l'entrée standard

Notes

Puissance : lecture au clavier

```
#include <stdio.h>
double puissance (int a, int b) {
  /* Rôle : retourne a^b (ou 1.0 si b < 0) */
  double z = 1.0;
  while (b > 0) {
    z *= a; /* z = z * a */
    b -= 1; /* b = b - 1; */
  }
  return z;
}
int main (void) {
  int x;
  int y;
  fprintf(stdout, "x = ");
  fscanf(stdin, "%d", &x);
  fprintf(stdout, "y = ");
  fscanf(stdin, "%d", &y);
  double p = puissance(x, y);
  fprintf(stdout, "x = %d, y = %d, x^y = %.2f \n", x, y, p);
  return 0;
}
```


Notes

Lecture au clavier

- Dans les précédentes versions, pour modifier les valeurs de `x` et de `y`, il fallait modifier le texte source du programme, le recompiler et l'exécuter
- En C, on peut demander à l'utilisateur des valeurs sur l'entrée standard
  - `gcc -Wall -pedantic -ansi foo.c`
  - `./a.out`

```
x = 3
y = 4
x = 3, y = 4, x^y = 81.00
```


Notes

Un autre exemple

Écrire sur la sortie standard ce qui est lu sur l'entrée standard (l'entrée et la sortie standard sont ouvertes par défaut)

En pseudo-langage

```
// c un caractère
lire(c)
tant que (non findefichier(entrée)) {
  afficher(c)
  lire(c)
}
```


Notes

En C

```c
#include <stdio.h>
int main (void) {
  char c;
  c = fgetc(stdin);
  while (c != EOF) {
    fputc(c, stdout);
    c = fgetc(stdin);
  }
  return 0;
}
```

En plus court

```c
#include <stdio.h>
int main (void) {
  char c;
  while ((c = fgetc(stdin)) != EOF) {
    fputc(c, stdout);
  }
  return 0;
}
```


Notes

Un autre exemple

Compter le nombre de caractères lus sur l'entrée standard et écrire le résultat sur la sortie standard

En pseudo-langage

```
// nb un entier
nb <- 0
tant que (non findefichier(entrée)) {
  nb <- nb + 1
}
afficher(nb)
```


Notes

Une première version

```c
#include <stdio.h>

/* Compte le nombre de caractères lus sur l'entrée standard jusqu'à une
    fin de fichier */
long compte (void); //déclaration de compte

int main (void) {
  fprintf(stdout, "nb de caractères = %ld\n", compte());
  return 0;
}

long compte (void) {
  long nb;
  nb = 0;
  while (getc(stdin) != EOF) {
    nb += 1;
  }
  return nb;
}
```


Notes

En C

```c
#include <stdio.h>

/* Compte le nombre de caractères lus sur l'entrée standard jusqu'à une
    fin de fichier */
extern long compte (void); //déclaration de compte

int main (void) {
  fprintf(stdout, "nb de caractères = %ld\n", compte());
  return 0;
}

long compte (void) {
  long nb;
  for (nb = 0; getc(stdin) != EOF; nb++) {
    /* Rien */
  }
  return nb;
}
```


Notes

Compilation séparée

On veut réutiliser le plus possible les codes existants

- On organise le code : on le sépare par thème, par module :
  - Les fonctions de maths
  - Les fonctions d'entrée/sortie (affichage/saisie)
  - ...
- On met un ensemble de code source de fonctions (**définition des fonctions**) dans le même fichier `.c`
- Pour donner accès aux autres à ces fonctions, on doit donner leur signature (type de retour, nombre de paramètres, type des paramètres) = **déclaration**. On met ces déclarations dans un fichier public le `.h`

Notes

Compilation séparée

fichier `math.h` : fichier de " déclarations "

```c
/* Retourne a^b (ou 1.0 si b < 0) */
double puissance (int, int);
```

fichier `math.c` : fichier de " définitions "

```c
#include "math.h"
double puissance (int a, int b) {
  double z = 1.0;
  while (b > 0) {
    z *= a; /* z = z * a */
    b--; /* b = b - 1 */
  }
  return z;
}
```


Notes

Compilation séparée

Fichier essai.c : fichier principal

```c
#include <stdio.h>
#include <stdlib.h>
#include "math.h"

int main (int argc, char *argv[]) {
  if (argc != 3) {
    fprintf(stderr, "usage: %s x y >= 0 (x^y)\n", argv[0]);
    return 1;
  }
  int x = atoi(argv[1]);
  int y = atoi(argv[2]);
  if (y < 0) {
    fprintf(stderr, "usage: %s x y >= 0 (x^y)\n", argv[0]);
    return 2;
  }
  printf("x = %d, y = %d, x^y = %.2f\n", x, y, puissance(x, y));
  return 0;
}
```


Notes

Compilation séparée

- Dans cette version, les valeurs de x et de y, seront données en arguments de la commande
  - `gcc -Wall -pedantic -ansi -c math.c`
  - `gcc -Wall -pedantic -ansi -c essai.c`
  - `gcc essai.o math.o`
  - `./a.out 3 4`
    `x = 3, y = 4, x^y = 81.00`
  - `./a.out 3`
    `usage: ./a.out x y >= 0 (x^y)`
  - `./a.out 3 -4`
    `usage: ./a.out x y >= 0 (x^y)`

Notes

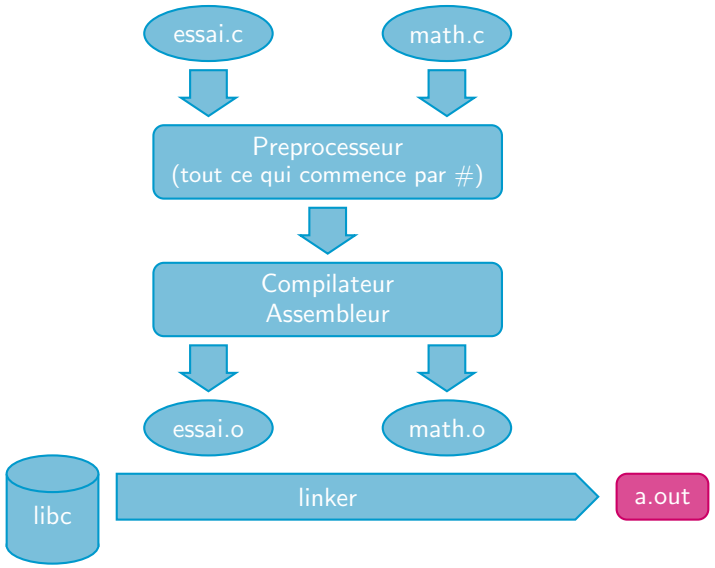
Déclarations et code

On va s'organiser un peu

- Il faut une déclaration avant utilisation
  - Prototypes mis dans un fichier : les `.h`
  - Code source des définitions : les `.c`
  - Certains fichiers sont déjà compilés (les objets) : les `.o`
  - On a des librairies (comme libc ou les maths) : les `.so`
- Il faut arriver à gérer tout cela :
  - On compile les `.c` avec les `.h`,
  - On ne veut pas tout recompiler quand un `.c` est modifié, mais une modification d'un `.h` peut avoir de l'importance
  - On utilise les autres `.o` et les `lib`
- Le gestionnaire : l'utilitaire `make` avec les makefile; ou bien votre interface de développement

Notes

Compilation



Notes

Eléments lexicaux

- Commentaires : `/* */`
- Identificateurs : suite de lettres non accentuées, de chiffres ou de souligné, débutant par une lettre ou un souligné
- Mots réservés
- Classes de variables : `auto`, `const`, `extern`, `register`, `static`, `volatile`
- Instructions : `break`, `case`, `continue`, `default`, `do`, `else`, `for`, `goto`, `if`, `return`, `switch`, `while`
- Types : `char`, `double`, `float`, `int`, `long`, `short`, `signed`, `unsigned`, `void`
- Constructeurs de types : `enum`, `struct`, `typedef`, `union`

Notes

Séquences d'échappement

- `\a` : Sonnerie
- `\b` : Retour arrière
- `\f` : Saut de page
- `\n` : Fin de ligne
- `\r` : Retour chariot
- `\t` : Tabulation horizontale
- `\v` : Tabulation verticale
- `\\` : Barre à l'envers
- `\?` : Point d'interrogation
- `\'` : Apostrophe
- `\"` : Guillemet
- `\o` `\oo` `\ooo` : Nombre octal
- `\xh` `\xhh` : Nombre hexadécimal

Notes

Constantes

Type entier en notation décimale, octale ou hexadécimale

int	unsigned int	long	long long ou __int64
123	12u	100L	1234LL
0173	0123u	125L	128LL
0x7b	0xAb3	0x12UL	0xFFFFFFFFFFFFFFFFLL

Type réel

float	double	long double
123f	123e0	123l
12.3F	12.3	12.3L

Notes

Constantes

Type caractère (char)

- Un caractère entre apostrophes
  - 'a'
  - '\141'
  - '\x61'
  - '\n'
  - '\0'
  - '\12'
- En C, un caractère est considéré comme un entier (conversion unaire)
  - char ca = 'a';
  - char ca = 97;
  - char ca = '\141';
  - char ca = '\x61';

Notes

Constantes

Type chaîne (`char *`) : chaîne placée entre guillemets

- `""`
- `"here we go"`
- `"une chaîne sur \`
  `deux lignes"`
- `"et"`
- `"une autre"`

Notes

Variable

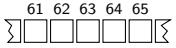
- Une variable est un nom auquel on associe une valeur que le programme peut modifier pendant son exécution
- Lors de sa déclaration, on indique son type
- **Il faut déclarer toutes les variables avant de les utiliser**
- On peut initialiser une variable lors de sa déclaration
- On peut préfixer toutes les déclarations de variables par `const` (jamais modifiés)

Notes

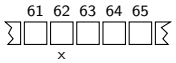
Variable

```
int x; // Réserve un emplacement pour un entier en mémoire
x = 10; // Écrit la valeur 10 dans l'emplacement réservé
```

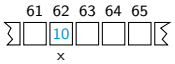
- Une variable est destinée à contenir une valeur du type avec lequel elle est déclarée
- Physiquement cette valeur se situe en mémoire



- `int x;`



- `x = 10;`



- `&x` : adresse de `x` en C : ici 62
  Adresse = numéro de la case mémoire

Notes

Types élémentaires

- Signé ou pas :
  - `unsigned` : non signé pas de négatif si n bits : $0 \dots (2^n - 1)$
  - `signed` : signé, négatifs, si n bits $-2^n - 1 \dots (2^{n-1} - 1)$
- Type entier :
  - `short`, signé par défaut en général sur 16 bits
  - `int`, signé par défaut, sur 32 bits en général
  - `long`, signé par défaut, sur 32 ou 64 bits
  - `long long` sur 64 bits
- Type réel :
  - `float`, signé, sur 32 bits en général
  - `double`, sur 32 ou 64 bits
  - `long double`, souvent sur 64 bits

Notes

Types élémentaires

- Type spécial : `void`
  - ne peut pas être considéré comme un type " normal "
- Type caractère :
  - `char`, signé par défaut : $-128 \cdots +128$
  - `unsigned char` : $0 \ldots 255$ parfois appelé `byte`
- **PAS de type booléen** : 0 représente le faux, et une valeur différente de 0 le vrai
- **ATTENTION** nombre de bits du type n'est pas dans le langage

Notes

Principe du C

- On écrit des valeurs dans des cases mémoires
- On lit des cases mémoire et on interprète le contenu de ce qu'on a lu
- `x` est un `int`. J'écris 65 dans `x`: `int x = 65;`
- Je lis la valeur de `x` : c'est 65
- Je place 65 dans un `char` : `char c = 65;`
- J'affiche le résultat: j'obtiens la lettre A
- J'ai interprété le résultat, la valeur n'a pas changé
- Pour afficher des caractères on utilise une table de conversion, dite table ASCII. Dans cette table la valeur 65 correspond à A

Notes

Table 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	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Notes

Principe du C

- On écrit des valeurs dans des cases mémoires
- On lit des cases mémoire et on interprète le contenu de ce qu'on a lu
- Ce qui est écrit est toujours écrit en binaire
- Codage des entiers (`int`, `long`, ...)
- Codages des flottants (`float`, `double`, ...)
- Ce n'est pas la même chose !
- Attention à l'interprétation

Notes

Représentation sur 32 bits (ou 64)

- Sur 32 bits, on peut représenter au plus 2^{32} informations différentes
- Si on représente des entiers on représente donc des nombres de 0 à 4 Milliards
- Comment représenter des nombres plus grands ?
- Comment représenter des nombres à virgule ?
- On va donner un sens différents aux bits

Notes

Représentation en base 2

- Système de numération (base 2, 10, 16)
- Représentation des entiers
- Représentation des nombres réels en virgule flottante

Notes

Bases 2, 10, 16

- $\forall$ base b, un nombre n s'écrit $n = \sum_{i=0}^{\infty} a_i b^i$
- Base 10 :
  - $a_i \in \{0, 1, 2, \ldots, 7, 8, 9\}$
  - $n = 1011_{10} = 1 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 1 \times 10^0 = 1011_{10}$
- Base 2 :
  - $a_i \in \{0, 1\}$
  - $n = 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}$
- Base 16 :
  - $a_i \in \{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$
  - $n = 1011_{16} = 1 \times 16^3 + 0 \times 16^2 + 1 \times 16^1 + 1 \times 16^0 = 4113_{10}$

Notes

Langage C Principe

Bases 2, 10, 16

Taille bornée des entiers stockés

Soit un entier m représenté sur n symboles dans une base b, on a $m \in [0, b^n - 1]$

Exemple

- sur 3 digits en décimal, on peut représenter les entiers [0, 999]
- sur 3 bits en binaire, on peut représenter les entiers [0, 7]
- sur 3 symboles en hexadécimal, on peut représenter les entiers [0, 4095]

Marie Pelleau Programmation C 2020-2021 48 / 57

Notes

Conversion vers la base 10

$\forall$ base b, un nombre n s'écrit $n = \sum_{i=0}^{\infty} a_i b^i$

Exemple

- $010100_2 = 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
  $= 16 + 4 = 20_{10}$
- $1111_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  $= 8 + 4 + 2 + 1 = 15_{10}$
- $012_{16} = 0 \times 16^2 + 1 \times 16^1 + 2 \times 16^0$
  $= 16 + 2 = 18_{10}$
- $1AE_{16} = 1 \times 16^2 + 10 \times 16^1 + 14 \times 16^0$
  $= 256 + 160 + 14 = 430_{10}$

Notes

Conversion de la base 10 à la base 2

- Comment à partir de n retrouver les a_i ? $n = \sum_{i=0}^{\infty} a_i 2^i$
- Divisions successives
  - Jusqu'à l'obtention d'un quotient nul
  - Lecture du bas vers le haut

$6 = 2 \times 3 + 0$
$3 = 2 \times 1 + 1$
$1 = 2 \times 0 + 1$

$6_{10} = 110_2$

- Tableau de puissance de 2
  - Parcourir le tableau des 2^i de gauche à droite
    - Si $n \geq 2^i$, alors mettre 1 dans la case 2^i et $n = n - 2^i$
    - Sinon mettre 0 dans la case 2^i

2^3	2^2	2^1	2^0
0	1	1	0

$6_{10} = 0110_2$

Notes

Conversion de la base 10 à la base 16

- $n = \sum_{i=0}^{\infty} a_i 16^i$
- Divisions successives
  - Jusqu'à l'obtention d'un quotient nul
  - Lecture du bas vers le haut

$$\begin{aligned} 687 &= 16\times 42 + 15 \\ 42 &= 16\times 2 + 10 \\ 2 &= 16\times 0 + 2 \end{aligned} \quad \begin{matrix} \mathrm{F} \\ \mathrm{A} \\ 2 \end{matrix} \quad \uparrow \quad 687_{10} = 2AF_{16}$$

- Tableau de puissance de 16
  - Parcourir le tableau des 16^i de gauche à droite
    - Si $n \geq 16^i$, alors mettre $n \div 16^i$ dans la case 16^i, et $n = n \mod 16^i$
    - Sinon mettre 0 dans la case 16^i

16^3	16^2	16^1	16^0
0	2	A	F

$\rightarrow$ $687_{10} = 02AF_{16}$

Notes

Conversions base 2, base 16

De la base 2 à la base 16

- Séparer le nombre binaire en quartet (de droite à gauche)
- Convertir chaque quartet en hexadécimal
- Exemple : 11011110001010000_2
- Séparation en quartet : 1 1011 1100 0101 0000
- Conversion : $1\ 1011\ 1100\ 0101\ 0000_2 = 1\ B\ C\ 5\ 0_{16}$

De la base 16 à la base 2

- Conversion de chaque symbole par un quartet
- Exemple : $AF23_{16}$
- En quartet : $A_{16} = 1010_2, F_{16} = 1111_2, 2_{16} = 0010_2, 3_{16} = 0011_2$
- Conversion : $AF23_{16} = 1010\ 1111\ 0010\ 0011_2$

Notes

Exemples

- $1010_2 =$
- $1101000_2 =$
- $12_{16} =$
- $3A_{16} =$
- $27_{10} =$
- $35_{10} =$

Notes

Réels en virgule flottante

- On représente un nombre avec une **mantisse** et un **exposant**, similairement à la notation scientifique :
  $1,234 \times 10^3 (= 1234)$
- La mantisse et l'exposant doivent être représentés sur un **nombre fixe de chiffres**
- Le **biais**, pour avoir des exposants négatifs

Exemple

- Mantisse sur 5 chiffres $\left(\frac{1}{10^0} + \frac{2}{10^1} + \frac{3}{10^2} + \frac{4}{10^3} + \frac{0}{10^4}\right)$ $m = 12340$
- Exposant sur 2 chiffres $(3 + 50)$ $e = 53$
- Biais de 50

  (12340|53)

Notes

Réels en virgule flottante

- $0,0005$ (5×10^{-4}) serait représenté par $m = 50000, e = 46$ $(50000|46)$
- Le plus grand nombre représentable est $(99999|99) = 9,9999 \times 10^{99} \times 10^{-50} = 9,9999 \times 10^{49}$
- Le plus petit (strictement positif) est $(00001|00) = 1.10^{-4} \times 10^{-50} = 10^{-54}$
- On ne peut pas représenter **exactement** 1,23456 sur cet exemple, puisque la mantisse ne peut avoir que 5 chiffres

Notes

Réels binaires en virgule flottante

- La représentation binaire en virgule flottante est analogue
- Par exemple une mantisse sur 5 bits, un exposant sur 3 bits et un biais d valant 3
  $(10110|110) = \left(1 + \frac{0}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{0}{2^4}\right) \times 2^{6-3} = 11$
  $1011_2 = 11_{10}, 1,0110_2 = 11_{10} \times 2^{-3}, (11 \times 2^{-3}) \times 2^3$
- La mantisse $b_0 b_1 b_2 \dots b_m$ représente le nombre **rationnel** $m = b_0 + \frac{b_1}{2} + \frac{b_2}{2^2} + \dots + \frac{b_m}{2^m}$
  (en forme normalisée, on a toujours $b_0 = 1$)
- L'exposant $x_n x_{n-1} \dots x_1 x_0$, représente l'**entier** $e = x_n \times 2^n + \dots + x_1 \times 2 + x_0$,
- La valeur représentée par le couple (m, e) est $m.2^{e-d}$

Notes

Réels binaires en virgule flottante

- La conversion de $0, xxxxx$ en binaire se fait en procédant par **multiplications successives**
- $0,375 = \frac{1}{4} + \frac{1}{8}$

$0,375 \times 2 = 0,75$
$0,75 \times 2 = 1,5$
$0,5 \times 2 = 1$

$0,375_{10} = 0,011_2$

Notes

Réels binaires en virgule flottante

- La norme IEEE 754 définit 2 formats (un peu plus compliqués que le modèle précédant)
  - Simple précision : mantisse sur 23 bits, exposant sur 8, $d = 127$
  - Double précision : mantisse sur 52 bits, exposant sur 11, $d = 1023$
- Les calculs se font en précision étendue : mantisse sur au moins 64 bits, exposant sur au moins 15

Notes

Réels binaires en virgule flottante

- Sur 32 bits, la représentation entière jusqu'à l'ordre de grandeur 4×10^9, la représentation flottante représente les nombres entre les ordres de grandeur 10^{-37} et 10^{37}
- Sur 64 bits, avec les entiers 1×10^{19}, avec les flottants 10^{-308} et 10^{308}
- Mais cette faculté d'exprimer de très petits ou de très grands nombres se paye par une approximation puisque seuls peuvent être **représentés exactement les nombres de la forme** $(b_0 + \frac{b_1}{2} + \frac{b_2}{2^2} + \cdots + \frac{b_m}{2^m}) \times 2^{e-d}$
- Sur n bits, on représente moins de nombres réels flottants différents que de nombres entiers !

Notes

Réels binaires en virgule flottante

- On a donc des erreurs d'arrondi : par exemple sur 32 bits, $1000 \times (\frac{1}{3000} + \frac{1}{3000} + \frac{1}{3000}) \neq 1$
- $0,3_{10}$ n'est pas exactement représentable en binaire !
  $0,3 \times 2 = 0,6$
  $0,6 \times 2 = 1,2$
  $0,2 \times 2 = 0,4$
  $0,4 \times 2 = 0,8$
  $0,8 \times 2 = 1,6$
  $0,6 \times 2$: on boucle

  $0,3_{10} = 0,0100110011001\ldots_2$
- 0,3 a une représentation finie en base 10 mais pas en base 2
- **Tous les nombres à virgule ne sont pas représentables**

Notes

Norme IEEE 754

- Le standard IEEE en format simple précision utilise 32 bits pour représenter un nombre réel x :
- $x = (-1)^s \times 2^{e-127} \times 1, m$
  - s est le bit de signe (1 bit)
  - e l'exposant (8 bits)
  - m la mantisse (23 bits)
- Pour la double et quadruple précision, le nombre de bits de la mantisse et de l'exposant, et donc le biais diffèrent

Notes

Notes