Introduction à l'informatique

Les formats

Jean-Christophe Dubacq

IUT de Villetaneuse

S1 2016

Plan

Plan

L'assemblage de types

Presque tous les langages de programmation utilisent une notion de *types de base* et de *composition de types*.

Types de base

Les langages ont presque tous un type *entier* et un type *flottant* (certains n'ont pas le premier). Il y a aussi souvent un type *booléen* et un type *caractère* qui désigne un caractère.
Ces types de base sont les éléments les plus

Assemblage

Les assemblages typiques sont les **assemblages par répétition** (structures de *tableau* (taille fixe), de vecteur ou de liste (taille arbitraire), mais aussi des **assemblages hétérogènes** de taille fixe.
Une dernière option est l'assemblage de type *variante* qui consiste à pouvoir mettre dans un même emplacement un type ou un autre. Certains langages ne spécialisent même pas les variables.

Exercices

Types simples ou composés

Q1 Identifiez dans les types suivants lesquels sont susceptibles d'êtres des types de base et lesquels sont plutôt des types construits par assemblage :

- Un nombre entier positif ou nul
- Un nombre complexe
- Un point dans l'espace
- Un nombre avec un très grand nombre de chiffres non fixé à l'avance
- Un intervalle
- Une date
- Un étudiant (nom, prénom, date de naissance)
- Un caractère
- Une chaîne de caractères

Une date

Q2 Décrivez à partir de quels éléments on peut composer une donnée qui représente un moment précis de la journée.

Q3 Discutez les éléments précis selon que l'on considère qu'un moment est pris à la seconde près ou beaucoup plus précis.

Éléments de taille variable

Il arrive que l'on veuille représenter une structure complexe qui compte un nombre variable d'autres structures.

Par exemple une chaîne de caractères !

- Problème de détection de la fin.
- Codage du nombre d'éléments, ou séquence de fin.

Lors de la construction de structures complexes à partir de structures élémentaires, on peut externaliser la structure de taille variable.

Une structure de taille fixe est plus facilement manipulable (notamment pour en faire des vecteurs ou des listes).

Représentation en mémoire

Représentation dans un fichier

- L'unité de base est l'octet
- Les données peuvent être indexées (on connaît le début de chaque donnée)
- Les données peuvent être typées (on connait le type de chaque donnée)
- Compromis entre taille occupée et résistance aux erreurs ou déchiffrabilité

Représentation en mémoire vive

- Les données simples ont souvent une représentation en mémoire qui occupe un nombre d'octets contigus fixe.
- Les données composites sont stockées par juxtaposition des données élémentaires qui les composent.
- Les données de taille variables sont *externalisées*. On les connaît alors par leur **adresse**, l'emplacement mémoire où elles sont stockées.

Plan

Du nombre de bits d'un processeur

- Un processeur manipule des données avec une taille fixe

Il y a parfois plusieurs tailles manipulables

- Historique :
  - Ordinateurs 8 bits : 1972–1985
  - Ordinateurs 16 bits : 1975–1990
  - Ordinateurs 32 bits : 1986–maintenant
  - Ordinateurs 64 bits : 1992/2003–maintenant
- Pour les flottants ou les données graphiques, il y a des circuits spécialisés qui ont des tailles différentes (plus grandes)
- Les données de taille supérieure doivent être manipulées en plusieurs opérations et ne sont pas des données simples

C'est la taille du *mot-machine*.

Données simples du C

- `char` (C2 8 bits)
- `short int` (C2 $\geq$16 bits)
- `int` (C2 $\geq$16 bits, usuellement 32)
- `long int` (C2 $\geq$32 bits)
- `long long int` (C2 $\geq$64 bits)
- `float` est l'IEEE754 simple précision (32 bits), `double` est la double précision (64 bits), `long double` est la précision étendue ou quadruple selon les processeurs, $\geq$80 bits).
- une *adresse* permet de désigner une autre donnée dans la mémoire. Leur taille a varié dans l'histoire de l'informatique.
- N'oubliez pas qu'un nombre peut cacher un champ de bits
- Des versions `unsigned` de toutes les longueurs d'entiers existent et permettent de choisir le codage NAT au lieu de C2.

Les modèles de programmation 32 et 64 bits

Besoin de garder la compatibilité des codes sources.
En C et en C++ les types changent de taille.

Modèle	Taille `short int`	Taille `int`	Taille `long`	Taille adresse	Taille `long long`	Exemples de Compilateurs
32 bits	16	32	32	32	—	
LP64	16	32	64	64	—	Tous sauf...
ILP64	16	64	64	64	—	Exceptions...
LLP64	16	32	32	64	64	Microsoft...

Alignement

Qu'est-ce que l'alignement ?

Les processeurs présentent des contraintes techniques pour l'adressage des données. Une opération sur une donnée de type simple en mémoire doit se faire avec une adresse multiple de sa taille.
Les types de taille supérieure au mot-machine sont de toute façon manipulée en morceaux indépendants.

Exemple

Sur une machine 32 bits, un `int` (4 octets) ne peut pas commencer à l'adresse `0x00000002`. Il commence soit à l'adresse `0x00000000`, soit `0x00000004`. Par contre, un `short int` de 16 bits pourra commencer à cette adresse.

L'alignement dans les données composées

Sauf exception, l'alignement doit être respecté pour toutes les composantes de la donnée composée. L'adresse de la donnée composée est l'adresse de la première composante, et chaque composante doit être alignée correctement vis-à-vis de sa taille.

Exercices

Stockage d'une date (suite)

Une date est composée des éléments suivants :

- Une année (disons de -2 milliards à +2 milliards)
- Un mois, un jour du mois
- Un fuseau horaire qui est une « adresse »
- Une heure, une minute (entiers)
- Un nombre de secondes qui est un flottant simple précision

Q4 Dites quels sont les types de base du C à utiliser pour coder cette information, d'après les limites connues de stockage pour chaque type. Utilisez les tailles les plus petites possibles.

Q5 Donnez leur noms à la fois dans un modèle 32 bits et un modèle 64 bits.

Q6 Si on avait voulu aller de -5 milliards à +5 milliards d'années, quel type aurait-on dû utiliser ?

Q7 Si les données sont dans l'ordre indiqué dans un type composé, précisez à quel moment les contraintes d'alignement provoquent des « trous » dans la structure.

Q8 Quelle est la taille totale de la structure (avec les trous) ?

Grand et petit-boutien (little/big-endian)

- Valeur `0x4A3B2C1D`, adresse `0x00000000` ?

Plusieurs représentations possibles :

- Contenu | 4A | 3B | 2C | 1D | petit-boutien
  Adresse | 03 | 02 | 01 | 00 | *(little-endian)*
- Contenu | 1D | 2C | 3B | 4A | grand-boutien
  Adresse | 03 | 02 | 01 | 00 | *(big-endian)*

- little-endian : 6502, x86, VAX.
- big-endian : Motorola 68000, SPARC, System/370.
- bi-endian : ARM, PowerPC (sauf G5), MIPS.
- Les bi-boutiens ont un mode par défaut (big-endian pour PowerPC, little-endian pour IA-64).

Grand et petit-boutien (little/big-endian)

- Valeur `0x4A3B2C1D`, adresse `0x00000000` ?

Plusieurs représentations possibles :

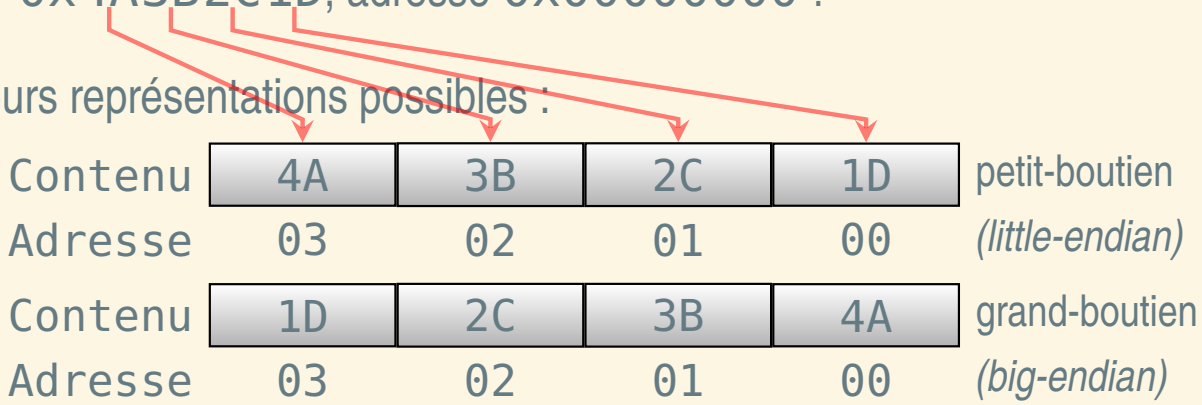
- `Contenu` | `4A` | `3B` | `2C` | `1D` — petit-boutien
  `Adresse` | `03` | `02` | `01` | `00` — *(little-endian)*
- `Contenu` | `1D` | `2C` | `3B` | `4A` — grand-boutien
  `Adresse` | `03` | `02` | `01` | `00` — *(big-endian)*

- little-endian : 6502, x86, VAX.
- big-endian : Motorola 68000, SPARC, System/370.
- bi-endian : ARM, PowerPC (sauf G5), MIPS.
- Les bi-boutiens ont un mode par défaut (big-endian pour PowerPC, little-endian pour IA-64).

Grand et petit-boutien (little/big-endian)

- Valeur `0x4A3B2C1D`, adresse `0x00000000` ?

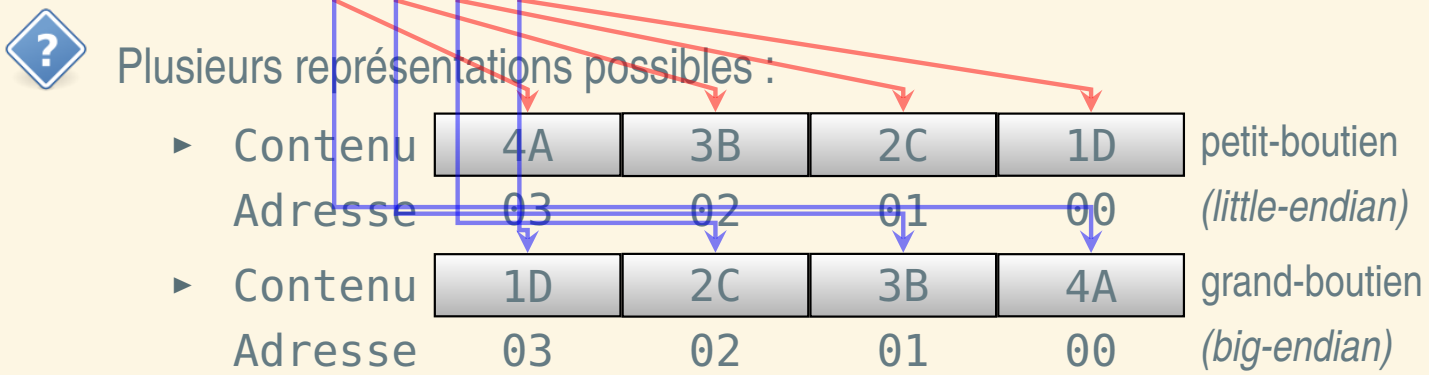
Plusieurs représentations possibles :



- little-endian : 6502, x86, VAX.
- big-endian : Motorola 68000, SPARC, System/370.
- bi-endian : ARM, PowerPC (sauf G5), MIPS.
- Les bi-boutiens ont un mode par défaut (big-endian pour PowerPC, little-endian pour IA-64).

Grand et petit-boutien (little/big-endian)

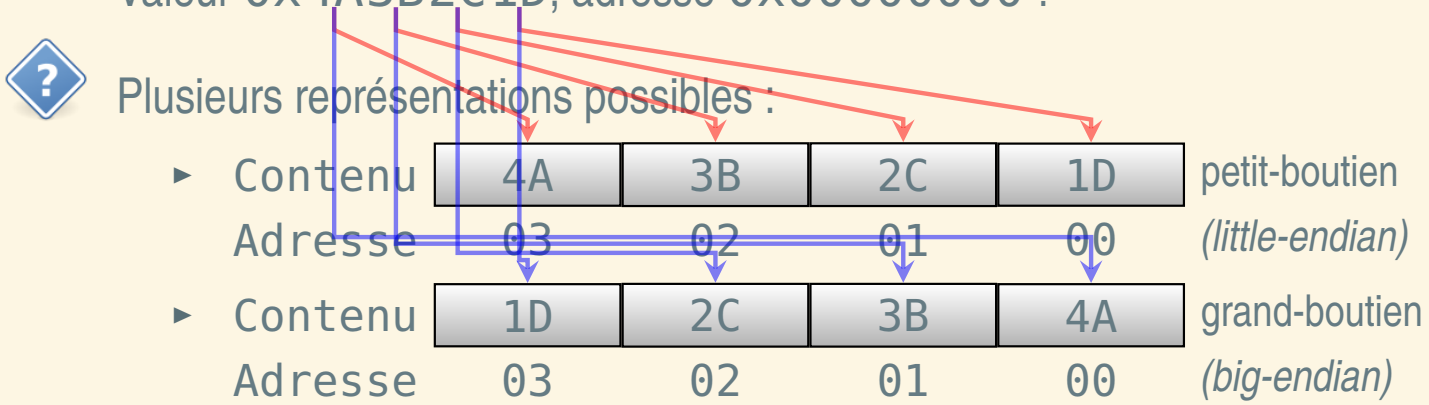
- Valeur `0x4A3B2C1D`, adresse `0x00000000` ?



- Plusieurs représentations possibles :
- little-endian : 6502, x86, VAX.
- big-endian : Motorola 68000, SPARC, System/370.
- bi-endian : ARM, PowerPC (sauf G5), MIPS.
- Les bi-boutiens ont un mode par défaut (big-endian pour PowerPC, little-endian pour IA-64).

Grand et petit-boutien (little/big-endian)

- Valeur `0x4A3B2C1D`, adresse `0x00000000` ?



- Plusieurs représentations possibles :
- little-endian : 6502, x86, VAX.
- big-endian : Motorola 68000, SPARC, System/370.
- bi-endian : ARM, PowerPC (sauf G5), MIPS.
- Les bi-boutiens ont un mode par défaut (big-endian pour PowerPC, little-endian pour IA-64).

Exercices

Stockage d'une date (suite)

Q9 On veut stocker la date du 24 décembre -4 à 21h45 dans un système 32 bits. Calculez les valeurs à stocker en hexadécimal (hormis l'adresse du fuseau horaire). Vous prendrez comme valeur d'adresse pour le fuseau horaire 0x12345678.

Q10 Écrivez, les uns après les autres, les octets qui composent cette date si on est dans un système 32 bits *big-endian*

Q11 Écrivez, les uns après les autres, les octets qui composent cette date si on est dans un système 32 bits *little-endian*

Plan

Exercices

Information intrinsèque

Q12 Si vous lancez une pièce de monnaie 20 fois en l'air, est-ce que vous avez plus de chance de tomber sur 20 fois face (F), 10 fois face-pile (FP), ou sur FPFPPFFFPPPPFPFFPPFP ?

Q13 Quelle est la quantité d'information contenue dans la suite binaire 110111001001 ? Et dans la suite binaire 000000000000 ? Et dans la suite binaire 010101010101 ?

Q14 Est-ce qu'on pourrait écrire certaines de ces suites de façons plus courtes ? Proposez-en.

La compression de données sans pertes

Complexité de Kolmogorov

La complexité d'une suite binaire est le programme le plus court qui permet de l'écrire (dans un langage adéquat).

Pour certaines suites, le programme le plus court est celui qui les contient. C'est lié à la notion d'aléatoire en *calculabilité*.

Cette complexité ne peut pas être calculée par un programme. Par contre, il est possible de trouver des programmes plus courts pour la plupart des données usuelles. C'est la **compression** !

Compression

La compression permet d'économiser de la place sur les disques et en mémoire.
Elle pénalise l'accès direct aux données. Elle se paye par des calculs de *décompression* pour y accéder.
En gagnant de la place sur le disque dur, elle permet d'accélérer le traitement des données, la lecture depuis le disque étant beaucoup plus lente que les calculs pour décompresser en général.

Les utilitaires classiques

L'archivage

L'archivage consiste à transformer toute une hiérarchie de fichiers en un fichier unique.

Ce n'est pas de la compression.

- Souvent couplé à une ou plusieurs méthodes de compression.
- Utilitaire `tar` sous Unix, programmes commerciaux `zip` ou `rar`.

Les formats classiques

- `gzip` utilise le codage de Lempel-Ziv (extension usuelle : `gz`)
- `zip` utilise le codage de Lempel-Ziv et de Huffman par dessus (extension usuelle : `zip`, mais aussi d'autres formats : `jar`, `odt`)
- `rar` (logiciel commercial) utilise le codage de Lempel-Ziv avec des techniques de prédiction PPM par dessus (extension usuelle : `rar`)

La compression RLE

Codage par plages

La compression RLE (pour *Run Length Encoding*) est une des compressions les plus simples : on transforme une suite de symboles en une suite de paires (nombre de symboles à répéter – symbole). Par exemple 00000011110011 se code **6**0**4**1**2**0**2**1.
Il peut mener à une chaîne plus grande que l'originale (par exemple pour 010101 : **1**0**1**1**1**0**1**1**1**0**1**1).
Le compte est souvent codé sur une taille fixe (un octet). Lorsqu'on dépasse, on peut découper en plusieurs *runs* de longueur maximale (sauf le dernier). Exemple : **255**0**45**0 pour un chaîne de 300 fois 0.
Lorsqu'on a seulement deux symboles, on peut ne noter que le compte, pas le symbole. On autorise alors les *runs* de 0 caractères, et on indique le caractère initial. Exemple : 0**255 0 45 1** pour 300 zéros suivis d'un 1.

Exercices

Compression RLE

Q15 Voilà des suites à compresser avec la compression RLE :

- 000011110010100001111111
- 111211211111221312211131122211113213211
- 0110100110010110

Protocole de fax

Q16 Un fax est une suite de points noirs et blancs. En utilisant les notations du RLE binaire données dans le cours, décrivez l'image à droite. On part en haut à gauche de l'image, on part à droite et on reprend à la fin de chaque ligne à la ligne suivante à gauche.