Éléments d'informatique – Cours 8. Types de données, représentations et entrées/sorties.

Pierre Boudes

15 novembre 2011

Représentation des données

- Représentation des entiers naturels
- Représentation des entiers relatifs
- Représentation des réels en virgule flottante

Types en C et entrées sorties associées

Conversions automatiques entre types

Démos et fin

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectaction
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, enregistrements
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique
  - préprocesseur du compilateur C (include, define)
  - Édition de lien
- Algorithmes élémentaires
- Méthodologie de résolution, manipulation sous linux

Représentation en binaire des données (rappel)

Représentation en binaire des données (rappel)

Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une quantité élémentaire d'information (oui ou non, ouvert ou fermé, etc.).

Représentation en binaire des données (rappel)

Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une quantité élémentaire d'information (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est consituée de bits.

Représentation des entiers naturels

- Dans une base donnée, un nombre entier positif est représenté de manière unique par une suite de chiffres de la base :

Représentation des entiers naturels

- Dans une base donnée, un nombre entier positif est représenté de manière unique par une suite de chiffres de la base :
  - En base 10, on écrit le nombre 109 comme la suite des chiffres 1, 0, 9 car :

$$109 = 1 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$$

Représentation des entiers naturels

- Dans une base donnée, un nombre entier positif est représenté de manière unique par une suite de chiffres de la base :
  - En base 10, on écrit le nombre 109 comme la suite des chiffres 1, 0, 9 car :

$$109 = 1 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$$

  - Et en base 2, le nombre 25 s'écrit comme la suite des bits 1, 1, 0, 0, 1 car :

$$\underline{11001} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Représentation des entiers naturels

- Dans une base donnée, un nombre entier positif est représenté de manière unique par une suite de chiffres de la base :
  - En base 10, on écrit le nombre 109 comme la suite des chiffres 1, 0, 9 car :

$$109 = 1 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$$

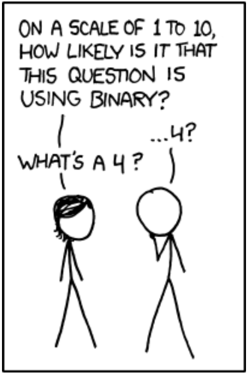
  - Et en base 2, le nombre 25 s'écrit comme la suite des bits 1, 1, 0, 0, 1 car :

$$\underline{11001} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

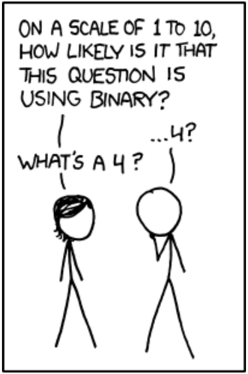
  - Plus généralement, la correspondance entre la représentation et le nombre est donnée par :

$$\underline{b_k \ldots b_0} = \sum_{i=0}^{k} b_i \times 2^i$$

Les blagues d'informaticiens (ici XKCD)

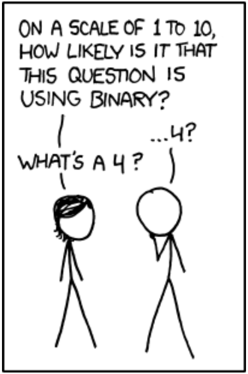


Les blagues d'informaticiens (ici XKCD)



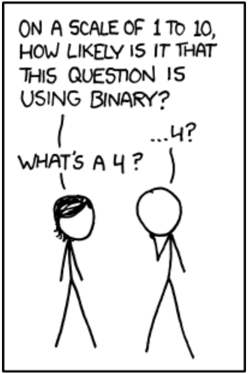
- Sur une échelle de 1 à 10, combien de chances y a t'il que cette question utilise le binaire ?

Les blagues d'informaticiens (ici XKCD)



- Sur une échelle de 1 à 10, combien de chances y a t'il que cette question utilise le binaire ?
- . . . 4 ?

Les blagues d'informaticiens (ici XKCD)



- Sur une échelle de 1 à 10, combien de chances y a t'il que cette question utilise le binaire ?
- . . . 4 ?
- Qu'est-ce qu'un 4 ?

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, . . . , n bits 2^n possiblités.

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, ..., n bits 2^n possiblités.
- Avec n bits on peut écrire les 2^n premiers nombres de 0 à $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. En général, sur ordinateur, n est fixé.

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, ..., n bits 2^n possiblités.
- Avec n bits on peut écrire les 2^n premiers nombres de 0 à $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. En général, sur ordinateur, n est fixé.
- L'addition se fait comme en base 10, c'est même encore plus facile :

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, . . . , n bits 2^n possiblités.
- Avec n bits on peut écrire les 2^n premiers nombres de 0 à $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. En général, sur ordinateur, n est fixé.
- L'addition se fait comme en base 10, c'est même encore plus facile :

$$
\begin{array}{rrlr}
 & \underline{11001} & (= & 25) \\
+ & \underline{11} & (= & 3) \\
\hline
 & \underline{11100} & (= & 28)
\end{array}
$$

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, ..., n bits 2^n possiblités.
- Avec n bits on peut écrire les 2^n premiers nombres de 0 à $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. En général, sur ordinateur, n est fixé.
- L'addition se fait comme en base 10, c'est même encore plus facile :

$$
\begin{array}{crcr}
 & \underline{11001} & (= & 25) \\
+ & \underline{11} & (= & 3) \\
\hline
 & \underline{11100} & (= & 28)
\end{array}
$$

- Dans une représentation de taille fixée (sur ordinateur), il y a un risque de débordement :

$$
\begin{array}{crcr}
 & \underline{11001} & (= & 25) \\
+ & \underline{111} & (= & 7) \\
\hline
 & \underline{00000} & (= & 0)
\end{array}
$$

- 1 bit représente 2 possibilités, 2 bits 4 possibilités, 3 bits 8 possibilités, . . . , n bits 2^n possiblités.
- Avec n bits on peut écrire les 2^n premiers nombres de 0 à $\sum_{i=0}^{n-1} 2^i = 2^n - 1$. En général, sur ordinateur, n est fixé.
- L'addition se fait comme en base 10, c'est même encore plus facile :

$$\begin{array}{rrlr} & \underline{11001} & (= & 25) \\ + & \underline{11} & (= & 3) \\ \hline & \underline{11100} & (= & 28) \end{array}$$

- Dans une représentation de taille fixée (sur ordinateur), il y a un risque de débordement :

$$\begin{array}{rrlr} & \underline{11001} & (= & 25) \\ + & \underline{111} & (= & 7) \\ \hline & \underline{00000} & (= & 0) \end{array} \qquad \begin{array}{rrlr} & \underline{11011} & (= & 27) \\ + & \underline{111} & (= & 7) \\ \hline & \underline{00010} & (= & 2) \end{array}$$

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010
  - On ajoute 1 11011

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010
  - On ajoute 1 11011
  - On obtient alors -5. L'addition avec 5 fait zéro !

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010
  - On ajoute 1 11011
  - On obtient alors -5. L'addition avec 5 fait zéro !
- Ça fonctionne bien car :
  - ajouter un nombre binaire avec le même nombre dont les bits ont étés inversés, donne $11\ldots1$.

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010
  - On ajoute 1 11011
  - On obtient alors -5. L'addition avec 5 fait zéro !
- Ça fonctionne bien car :
  - ajouter un nombre binaire avec le même nombre dont les bits ont étés inversés, donne $11\ldots1$.
  - À nombre de bits fixé, ajouter 1 donne zéro, par débordement.

Représentation des entiers relatifs

- Pour représenter les entiers relatifs, on peut réserver 1 bit au stockage du signe de l'entier (0 positif, 1 négatif) et représenter en base 2 sa valeur absolue sur les bits restants.
- Ça se fait parfois, mais il y a deux inconvénients :
  - Il y a deux représentation du zéro 10000 et 00000
  - L'addition doit être modifiée
- Alternative : la représentation en *complément à deux*.
  - Pour coder -5, on commence par coder 5 : 00101
  - On inverse les bits 11010
  - On ajoute 1 11011
  - On obtient alors -5. L'addition avec 5 fait zéro !
- Ça fonctionne bien car :
  - ajouter un nombre binaire avec le même nombre dont les bits ont étés inversés, donne $\underline{11\ldots1}$.
  - À nombre de bits fixé, ajouter 1 donne zéro, par débordement.
- Avec n bits on représente les nombres de -2^{n-1} à $2^{n-1}-1$.
- Remarque : le premier bit reste un bit de signe.

Représentation des réels en virgule flottante

Représentation des réels en virgule flottante

- La représentation informatique usuelle des réels s'inspire de la notation scientifique :

$$\pi = 3,141592653589793 \qquad \text{(pi)}$$
$$-700 \text{ milliards} = -7 \times 10^{11} \qquad \text{(Paulson)}$$
$$h = 6,626068 \times 10^{-34} \qquad \text{(Planck)}$$
$$\text{Univers} = 1 \times 10^{80} \qquad \text{(Atomes)}$$

Représentation des réels en virgule flottante

- La représentation informatique usuelle des réels s'inspire de la notation scientifique :

$$\pi = 3,141592653589793 \qquad \text{(pi)}$$

$$-700 \text{ milliards} = -7 \times 10^{11} \qquad \text{(Paulson)}$$

$$h = 6,626068 \times 10^{-34} \qquad \text{(Planck)}$$

$$\text{Univers} = 1 \times 10^{80} \qquad \text{(Atomes)}$$

- Les bits sont séparés en :
  - bit de signe (1 bit)
  - mantisse (53 bits)
  - exposant (11 bits)

Représentation des réels en virgule flottante

- La représentation informatique usuelle des réels s'inspire de la notation scientifique :

$$\pi = 3,141592653589793 \qquad \text{(pi)}$$

$$-700 \text{ milliards} = -7 \times 10^{11} \qquad \text{(Paulson)}$$

$$h = 6,626068 \times 10^{-34} \qquad \text{(Planck)}$$

$$\text{Univers} = 1 \times 10^{80} \qquad \text{(Atomes)}$$

- Les bits sont séparés en :
    - bit de signe (1 bit)
    - mantisse (53 bits)
    - exposant (11 bits)
- En double précision (64 bits) :
    - exposant : entre 10^{-308} et 10^{308} (environ).
    - mantisse : 16 chiffres décimaux (environ).

Représentation des réels en virgule flottante

- La représentation informatique usuelle des réels s'inspire de la notation scientifique :

$$\pi = 3,141592653589793 \qquad \text{(pi)}$$
$$-700 \text{ milliards} = -7 \times 10^{11} \qquad \text{(Paulson)}$$
$$h = 6,626068 \times 10^{-34} \qquad \text{(Planck)}$$
$$\text{Univers} = 1 \times 10^{80} \qquad \text{(Atomes)}$$

- Les bits sont séparés en :
  - bit de signe (1 bit)
  - mantisse (53 bits)
  - exposant (11 bits)
- En double précision (64 bits) :
  - exposant : entre 10^{-308} et 10^{308} (environ).
  - mantisse : 16 chiffres décimaux (environ).
- Infini positif, infini négatif.
- NaN : not a number.

Types en C et entrées/sorties associées

Types en C et entrées/sorties associées

- Type des entiers relatifs `int` :
  - Déclaration et initialisation : `int n = -23;`.
  - Représentation en complément à deux.
  - E/S : `%d`.

Types en C et entrées/sorties associées

- Type des entiers relatifs `int` :
  - Déclaration et initialisation : `int n = -23;`.
  - Représentation en complément à deux.
  - E/S : `%d`.
- Type des caractères `char` :
  - Déclaration et initialisation : `char c = 'A';`.
  - Représentation sur 8 bits, ASCII, ISO-8859-x, UTF-8.
  - E/S : `%c`.

Types en C et entrées/sorties associées

- Type des entiers relatifs `int` :
  - Déclaration et initialisation : `int n = -23;`.
  - Représentation en complément à deux.
  - E/S : `%d`.
- Type des caractères `char` :
  - Déclaration et initialisation : `char c = 'A';`.
  - Représentation sur 8 bits, ASCII, ISO-8859-x, UTF-8.
  - E/S : `%c`.
- Type des réels `double` :
  - Déclaration et initialisation : `double x = 3.14e-3;`.
  - Représentation en virgule flottante sur 64 bits.
  - E/S : `%lg` (mais plutôt %g avec `printf`).
  - Attention : toujours mettre le point (équivalent anglais de la virgule) pour les constantes réelles (`1.0`).

Entiers

```
int n;
...
printf("Entrer un nombre entier\n");
```

Entiers

```
int n;
...
printf("Entrer un nombre entier\n");
scanf("%d", &n);
```

Entiers

```
int n;
...
printf("Entrer un nombre entier\n");
scanf("%d", &n);
```

Réels

```
double x;
...
printf("Entrer un nombre reel\n");
scanf("%lg", &x);
```

Entiers

```
int n;
...
printf("Entrer un nombre entier\n");
scanf("%d", &n);
```

Réels

```
double x;
...
printf("Entrer un nombre reel\n");
scanf("%lg", &x);
printf("Vous avez saisi : %g\n", x);
```

Caractères

```
char c;
...
printf("Entrer un caractère\n");
scanf("%c", &c);
```

Caractères

```
char c;
...
printf("Entrer un caractère\n");
scanf("%c", &c);
```

Attention : mieux vaut utiliser `scanf(" %c", &c);` (voir démo)

Caractères

```
char c;
...
printf("Entrer un caractère\n");
scanf("%c", &c);
```

Attention : mieux vaut utiliser `scanf(" %c", &c);` (voir démo)

Chaînes de caractères

```
char nom[64];
...
printf("Entrer votre nom\n");
scanf("%s", nom);
```

Caractères

```
char c;
...
printf("Entrer un caractère\n");
scanf("%c", &c);
```

Attention : mieux vaut utiliser `scanf(" %c", &c);` (voir démo)

Chaînes de caractères

```
char nom[64];
...
printf("Entrer votre nom\n");
scanf("%s", nom);
printf("Vous avez saisi : %s\n", nom);
```

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)
- Avec changement de représentation :
  - char ou entiers vers réels
  - réels vers entiers ou char

Démos et fin