Cours Langage C/C++
Annexe sur les variables

Thierry Vaira

BTS IRIS Avignon

tvaira@free.fr © v0.1

Les types entiers

- `bool` : false ou true → booléen (seulement en C++)
- `unsigned char` : 0 à 255 ($2^8 - 1$) → entier très court (1 octet ou 8 bits)
- `[signed] char` : -128 (-2^7) à 127 ($2^7 - 1$) → idem mais en entier relatif
- `unsigned short [int]` : 0 à 65535 ($2^{16} - 1$) → entier court (2 octets ou 16 bits)
- `[signed] short [int]` : -32768 (-2^{15}) à +32767 ($2^{15} - 1$) → idem mais en entier relatif
- `unsigned int` : 0 à 4.295e9 ($2^{32} - 1$) → entier sur 4 octets ; **taille "normale" actuelle**
- `[signed] int` : -2.147e9 (-2^{31}) à +2.147e9 ($2^{31} - 1$) → idem mais en entier relatif
- `unsigned long [int]` : 0 à 4.295e9 → entier sur 4 octets ou plus ; sur PC identique à "int" (hélas...)
- `[signed] long [int]` : -2.147e9 à -2.147e9 → idem mais en entier relatif
- `unsigned long long [int]` : 0 à 18.4e18 ($2^{64} - 1$) → entier (très gros !) sur 8 octets sur PC
- `[signed] long long [int]` : -9.2e18 (-2^{63}) à -9.2e18 ($2^{63} - 1$) → idem mais en entier relatif

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Les types à virgule flottante

- `float` : Environ 6 chiffres de précision et un exposant qui va jusqu'à $\pm 10^{\pm 38}$ $\rightarrow$ Codage IEEE754 sur 4 octets
- `double` : Environ 10 chiffres de précision et un exposant qui va jusqu'à $\pm 10^{\pm 308}$ $\rightarrow$ Codage IEEE754 sur 8 octets
- `long double` $\rightarrow$ Codé sur 10 octets

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Les limites des nombres

- Le fichier `limits.h` contient, sous forme de constantes ou de macros, les limites concernant le codage des entiers et le fichier `float.h` contient celles pour les "floattants".

Exemple : les limites des nombres en C

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

int main() {
   printf("*** Limites pour les entiers ***\n");
   printf("Nombre de bits dans un char : %d bits\n", CHAR_BIT);
   printf("%d <= char <= %d\n", CHAR_MIN, CHAR_MAX);
   printf("0 <= unsigned char <= %u\n", UCHAR_MAX);
   printf("%d <= int <= %d\n", INT_MIN, INT_MAX);
   printf("0 <= unsigned int <= %u\n", UINT_MAX);
   printf("%ld <= long <= %ld\n", LONG_MIN, LONG_MAX);
   printf("0 <= unsigned long <= %lu\n", ULONG_MAX);
   printf("\n*** Limites pour les réels ***\n");
   printf("%e <= float <= %e\n", FLT_MIN, FLT_MAX);
   printf("%e <= double <= %e\n", DBL_MIN, DBL_MAX);
   printf("%Le <= long double <= %Le\n", LDBL_MIN, LDBL_MAX);
   return 0;
}
```

Exemple : exécution

```
*** Limites pour les entiers ***
Nombre de bits dans un char : 8 bits
-128 <= char <= 127
0 <= unsigned char <= 255
-2147483648 <= int <= 2147483647
0 <= unsigned int <= 4294967295
-2147483648 <= long <= 2147483647
0 <= unsigned long <= 4294967295

*** Limites pour les réels ***
1.175494e-38 <= float <= 3.402823e+38
2.225074e-308 <= double <= 1.797693e+308
3.362103e-4932 <= long double <= 1.189731e+4932
```

- Première loi : Le type d'une variable spécifie sa taille (de sa représentation en mémoire) et ses limites. Elle peut donc **déborder** (*overflow*).

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Exemple : débordons !

```c
#include <stdio.h>

int main()
{
   int entier = 2147483647; // la dernière valeur positive représentable pour un int
   unsigned char octet = 255 ; // la dernière valeur représentable pour un char soit 1111 1111

   printf("La variable entier a pour valeur max %d car sa taille est de %d octet(s)\n", entier, sizeof(int)
         );
   printf("La variable octet a pour valeur max %lu car sa taille est de %d octet(s)\n", octet, sizeof(
         unsigned char));

   /* Tentons l'impossible */
   entier = entier + 1; // peut s'écrire aussi : entier += 1;
   octet = octet + 1; // peut s'écrire aussi : octet += 1;

   printf("La variable entier + 1 a maintenant pour valeur %d !\n", entier);
   printf("La variable octet + 1 a maintenant pour valeur %lu !\n", octet);

   return 0;
}
```

Exemple : exécution

```
La variable entier a pour valeur max 2147483647 car sa taille est de 4 octet(s)

La variable octet a pour valeur max 255 car sa taille est de 1 octet(s)

La variable entier + 1 a maintenant pour valeur -2147483648 !

La variable octet + 1 a maintenant pour valeur 0 !
```

- Conclusion : le débordement est souvent une source d'erreur (*bug*) dans les programmes.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Le type char

Attention : Le binaire, le décimal et l'hexadécimal ne sont qu'une représentation numérique d'une même valeur.

Exemple : c'est quoi un caractère ?

```
#include <stdio.h>
void char2bit(char octet) {
   int i, rang = 0x80; // soit 1000 0000
   for(i = 0; i < 8; i++) {
      if(octet & (rang >> i)) // >> décalage à droite de i bits
            printf("1");
      else  printf("0");
   }
   printf("\n");
}

int main() {
   char lettre = 'A'; // le caractère ASCII 'A'
   printf("La variable lettre représente le caractère %c\n", lettre);
   printf("La variable lettre a pour valeur %d ou 0x%02X\n", lettre, lettre);
   printf("La variable lettre de type char a pour taille : %d octet(s)\n", sizeof(char));
   printf("\nMais sa valeur binaire dans la machine est tout simplement : "); char2bit(lettre);
   lettre = lettre + 1; // etc ...
   return 0;
}
```

C'est quoi un caractère ? Exécutons le programme d'essai :

```
La variable lettre représente le caractère A
La variable lettre a pour valeur 65 ou 0x41
La variable lettre de type char a pour taille : 1 octet(s)
Mais sa valeur binaire dans la machine est tout simplement : 01000001

lettre = lettre + 1;
La variable lettre représente le caractère B
La variable lettre a pour valeur 66 ou 0x42
La variable lettre de type char a pour taille : 1 octet(s)
Mais sa binaire dans la machine est tout simplement : 01000010
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Complément à 2

- Les `signed char/int` sont implantés en **complément à 2** pour intégrer **un bit de signe** pour représenter les entiers relatifs.
- Tout d'abord, on utilise le bit de poids fort (bit le plus à gauche) du nombre pour représenter son signe (égal à 1 et il est négatif et sinon il est positif).
- Il reste donc pour représenter le nombre : taille en bits - 1.
- Les nombres positifs sont représentés normalement, en revanche les nombres négatifs sont obtenus de la manière suivante :
  - On inverse les bits de l'écriture binaire de sa valeur absolue (opération binaire NON), on fait ce qu'on appelle le complément à un
  - On ajoute 1 au résultat (les dépassements sont ignorés).

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Complément à 2 : Exemple

- Pour coder le nombre (-4), on fera :
  - On prend le nombre positif 4 : `00000100`
  - On inverse les bits : `11111011`
  - On ajoute 1 pour obtenir le nombre -4 : `11111100`
- On peut remarquer que le bit de signe s'est positionné automatiquement à 1 pour indiquer un nombre négatif.
- La représentation en complément à 2 a été réténue car elle permet de d'effectuer les opérations arithmétiques usuelles naturellement. On peut vérifier que l'opération **3 + (-4)** se fait sans problème : `00000011 + 11111100 = 11111111`
- Le complément à deux de `11111111` est `00000001` soit 1 en décimal, donc avec le bit de signe à 1, ont obtient **(-1)** en décimal.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Exemple : complémentons alors !

```
#include <stdio.h>

void char2bit(char octet);

int main()
{
   char valeurNegative = -4;

   printf("La variable valeurNegative a pour valeur %d ou 0x%hhX\n",
        valeurNegative, valeurNegative);
   printf("\nSa représentation binaire en complément à 2 dans la machine est : "
        ); char2bit(valeurNegative);

   return 0;
}
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

On vérifie en exécutant le programme d'essai

```
La variable valeurNegative a pour valeur -4 ou 0xFC

Sa représentation binaire en complément à 2 dans la machine est : 1 1111100
```

Le standard IEEE 754

- Le **standard IEEE 754 définit les formats de représentation des nombres à virgule flottante** de type `float/double` (signe, mantisse, exposant, nombres dénormalisés) et valeurs spéciales (infinis et NaN), un ensemble d'opérations sur les nombres flottants et quatre modes d'arrondi et cinq exceptions (cf. fr.wikipedia.org/wiki/IEEE_754).
- La version 1985 de la norme IEEE 754 définit **4 formats** pour représenter des nombres à virgule flottante :
  - simple précision (32 bits : 1 bit de signe, 8 bits d'exposant (-126 à 127), 23 bits de mantisse, avec bit 1 implicite),
  - simple précision étendue (>= 43 bits, obsolète, implémenté en pratique par la double précision),
  - double précision (64 bits : 1 bit de signe, 11 bits d'exposant (-1022 à 1023), 52 bits de mantisse, avec bit 1 implicite),
  - double précision étendue (>= 79 bits, souvent implémenté avec 80 bits : 1 bit de signe, 15 bits d'exposant (-16382 à 16383), 64 bits de mantisse, sans bit 1 implicite).

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Mantisse et partie significative

- Le compilateur `gcc` (pour les architectures compatible Intel 32 bits) utilise le format **simple précision** pour les variables de type `float`, **double précision** pour les variables de type `double`, et la double précision ou la double précision étendue (suivant le système d'exploitation) pour les variables de type `long double`.
- La **mantisse est la partie décimale** de la partie significative, **comprise entre 0 et 1**.
- Donc **1+mantisse** représente la **partie significative**.
- Elle se code (attention on est à droite de la virgule) :
  - pour le premier bit avec le poids 2^{-1} soit `0,5`,
  - puis 2^{-2} donc `0,25`,
  - 2^{-3} donnera `0,125`, ainsi de suite ...
- Par exemple : `,010` donnera `,25`.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Interprétation d'un nombre à virgule flottante (1/2)

- L'interprétation d'un nombre est donc :
  $valeur = signe\ x\ 1 + mantisse\ x\ 2^{(exposant - decalage)}$
- Avec : $decalage = 2^{(e-1)} - 1$ (avec e sur 8 bits, on aura un décalage de 127)
- On va coder le nombre décimal -**118,625** en utilisant le mécanisme IEEE 754 :
  - C'est un nombre négatif, le signe est donc "1".
  - Puis on écrit le nombre (sans le signe) en binaire : `118 = 1110110` et `,625 = ,101` soit `1110110,101`.
  - Ensuite, on doit décaler la virgule vers la gauche, de façon à ne laisser qu'un 1 sur sa gauche, il faut `6 décalages` : `1110110,101` (bin) = $1,110110101\ x\ 2^6$.

Interprétation d'un nombre à virgule flottante (2/2)

- Donc :
  - La **mantisse** est la partie à droite de la virgule, remplie de 0 vers la droite pour obtenir **23 bits** en simple précision.
  - Cela donne 110 1101 0100 0000 0000 0000 (on ne met pas le 1 avant la virgule car il est implicite).
  - L'**exposant** est égal à **6**, et nous devons le convertir en binaire et le décaler.
  - Pour le format 32-bits IEEE 754, le décalage est $2^{(8-1)} - 1 = 127$. Donc $6 + 127 = 133 = 1000\ 0101$.
- On a donc -**118,625** = 1 10000101 11011010100000000000000

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Exemple : observons au microscope un float

```c
#include <stdio.h>

void float2bit(float reel); // affiche un float en binaire

int main()
{
   float reel = -118.625; // un exemple de réel

   printf("La variable reel a pour valeur %f ou %e\n", reel, reel);
   printf("La variable reel a pour représentation (en hexa) : %a\n", reel); //
        pour %a cf. man 3 printf
   printf("La variable reel de type float a pour taille : %d octet(s)\n", sizeof
        (float));

   printf("\nSa représentation IEEE754 en binaire dans la machine (signe
        exposant mantisse) :\n");
   float2bit(reel);

   return 0;
}
```

On vérifie en exécutant le programme d'essai

```
La variable reel a pour valeur -118.625000 ou -1.186250e+02

La variable reel a pour représentation (en hexa) : -0x1.da8p+6

La variable reel de type float a pour taille : 4 octet(s)

Sa représentation IEEE754 en binaire dans la machine (signe exposant mantisse) :
1 10000101 11011010100000000000000
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Décodage d'un nombre à virgule flottante

- Par exemple pour le nombre binaire (0 01111100 01000000000000000000000), on décode :
  - **1 bit** de **signe** : 0 ici donc **positif** (+)
  - **8 bits** d'**exposant** (donc un décalage de 127) : donc **e** vaut 124 - 127 = -3,
  - **23 bits** pour la **mantisse** : 1+mantisse représente la partie significative qui est 1,01 (en binaire) soit 1,25 en décimal ($1x2^0 + 0x2^-1 + 1x2^-2$) ou **1 + 0,25**
- La représentation sera :
  $valeur = signe\ x\ 1 + mantisse\ x\ 2^{(exposant-decalage)}$
- Le nombre représenté sur **32 bits** était donc : $+1,25x2^{-3}$ soit **0,15625**. Vous pouvez vérifier avec le programme d'essai fourni ci-dessus.

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Conséquence

- La représentation (signé, à virgule flottante, ...) associée à un type (signed int, float, ...) va avoir des conséquences importantes.
- Par exemple, vous devez maintenant comprendre qu'il est tout à fait **IMPOSSIBLE** pour la machine d'additionner un `int` avec un `float` : `1 + 0.5 ; // aie !`
- En effet, la machine ne les code pas de la même manière en binaire (on vient de le voir) et ne peut donc les additionner (ni faire une autre opération arithmétique). La machine ne sait faire que des opérations arithmétiques entre même type.
- Que fait la machine alors ? C'est tout simple : elle pratique la **conversion de type** !
  - $1 + 0.5 \rightarrow$ je ne peux pas donc je convertis `1` (`int`) en `1.0` (`float`) car je suis une machine maligne
  - $1.0 + 0.5 \rightarrow$ maintenant je peux faire l'addition de deux `float` car je suis une machine trop forte

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon