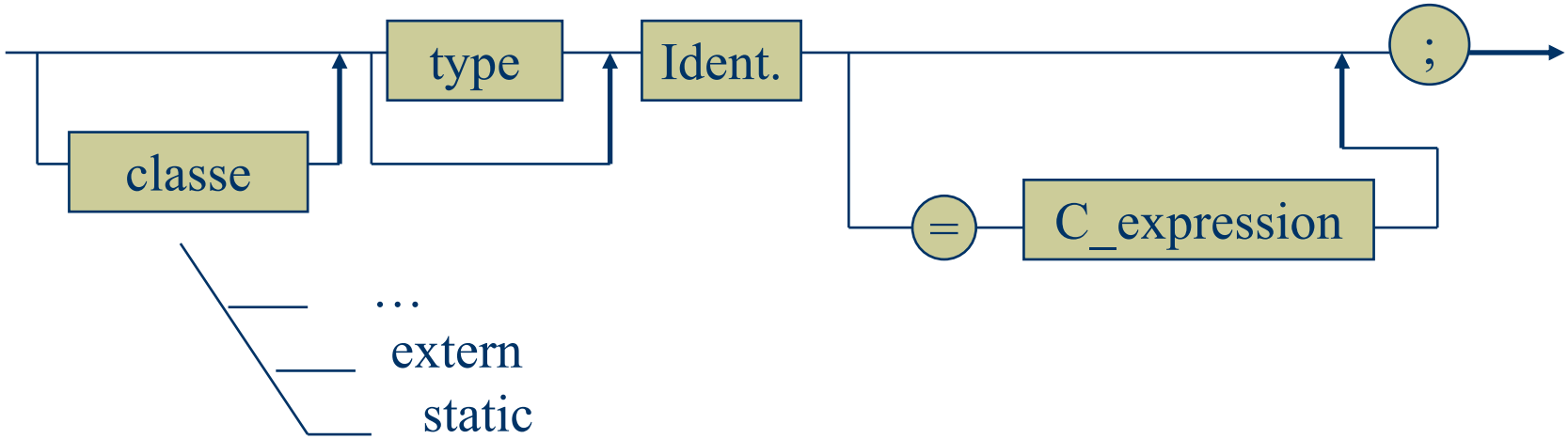
Les types de base

- Déclaration de variables
- Numération
- Numération : nombre négatifs
- Types simples : première partie
- Numération : nombres réels
- Types simples : deuxième partie
- Opérateurs arithmétiques et logiques
- Conversions
- Définitions de type

Les types de base
Déclaration de variables

Forme simple :




```
#define PI 3.14159
static int NbreDObjets = 0;
extern float Angle = PI/2;
```

Numération
Représentation

Le système décimal :

- système de numération à base 10
- symboles utilisés : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Ecriture d'un nombre décimal quelconque :

$N = a_n.10^n + a_{n-1}.10^{n-1} + \ldots + a_0.10^0$

$123 = 1.10^2 + 2.10^1 + 3.10^0$

Système à base quelconque :

$N = a_n.b^n + a_{n-1}.b^{n-1} + \ldots + a_0.b^0$ avec $0 <= a_i < b$

ou encore $N = a_n.a_{n-1} \ldots a_0$

Numération , Représentation
Cas du système binaire

symboles autorisés : 0, 1

$N = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_0 2^0$ avec $0 <= a_i < =1$

Exemple : $1010_2 = 1.2^3+0.2^2+1.2^1+0.2^0$ en base 10 = 10_{10}

En interne, toutes les informations sont codées sous forme de bits et plus particulièrement par l'intermédiaire de mots composés d'un certain nombre d'octets contenant eux même 8 bits.

Mot binaire de n bits permet de représenter des entiers naturels de 0 à 2^n-1

Sur un octet : 0 .. 2^8 -1 à savoir 0 .. 255

Sur 2 octets : 0 .. 2^{16}-1 à savoir 0 .. 65 535

Numération, Représentation des nombres négatifs (1)

Idée de base : utiliser le premier bit comme bit de signe.

0 pour un nombre positif

1 pour un nombre négatif

→ Sur un mot de 8 bits, on n'utilise que 7 bits pour coder un nombre positif : -2^7-1 .. 2^7-1 à savoir -127 .. 127

Convention : 1000 0000 = -2^7 = -128 ceci afin d'avoir un zéro unique

Un nombre entier codé sur 16 bits permet de représenter un nombre Appartenant à l'intervalle -32 768 .. 32767

Numération, Représentation des nombres négatifs (2)

Utilisation de la technique du complément à 2

Complément à 1 auquel on ajoute 1

Exemple : 131_{10} à savoir $1000\ 0011_2$

On retire 1 $0000\ 0001$

$1000\ 0010$

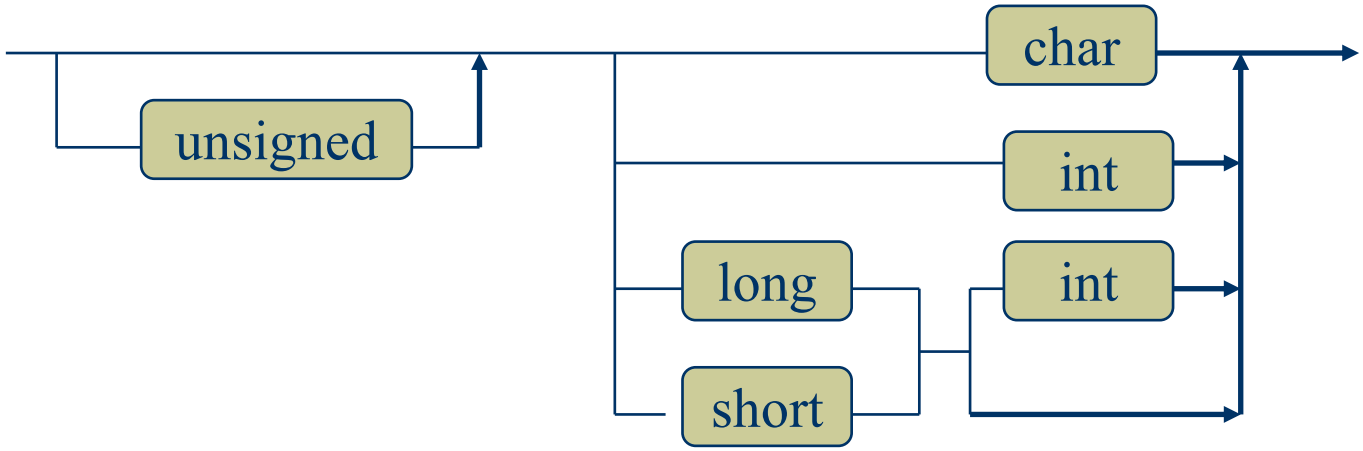
Complément à 1 donne $0111\ 1101$ soit 125_{10}

Puisque le bit de poids fort de 131 en binaire est 1

131 en binaire représente l'entier -125

Les types de base
Types simples (1)

Type entier

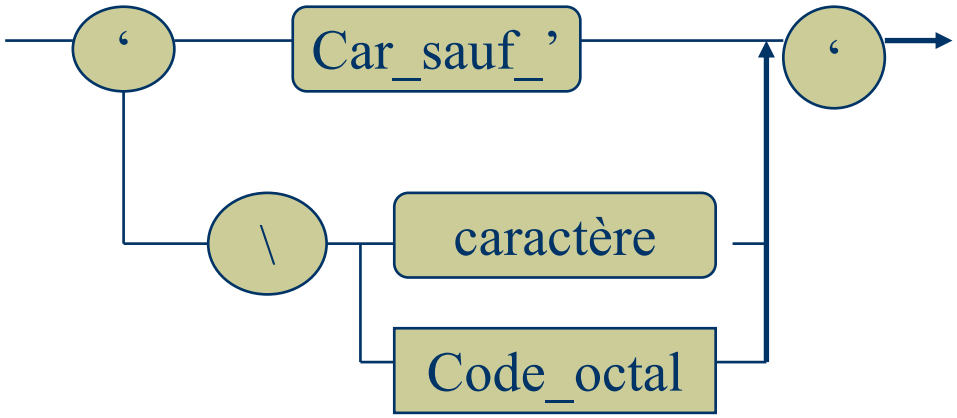


Les types de base
Types simples (2)

Type	Taille	Rang
unsigned char	1 octet	0 à 255
char	1 octet	-128 à 127
short	2 octets	-32 768 à 32 767
unsigned short	2 octets	0 à 65 535
unsigned int	4 octets	0 à 4 294 967 295
int	4 octets	-2 147 483 648 à 2 147 483 647
unsigned long	4 octets	0 à 4 294 967 295
long	4 octets	-2 147 483 648 à 2 147 483 647
unsigned long long	8 octets	0 à 18 446 744 073 709 551 615
long long	8 octets	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Les types de base
Types simples (3)

Constante_char



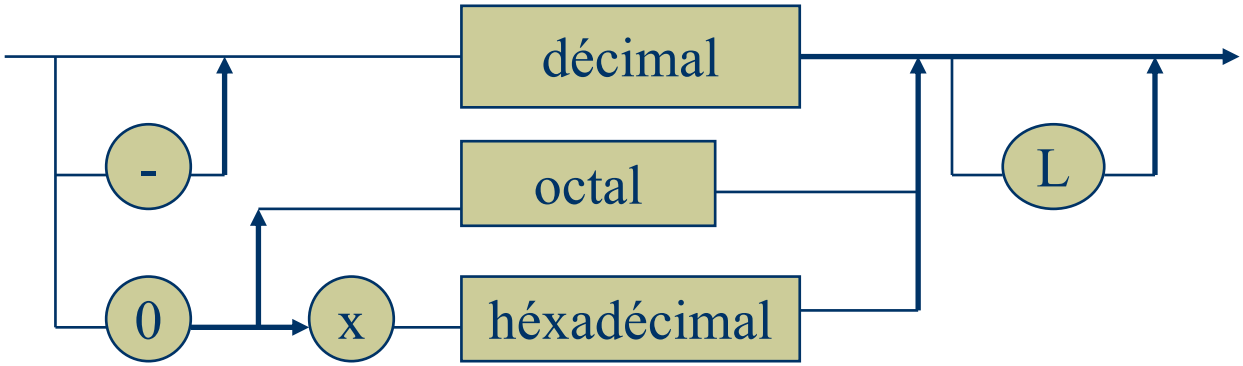
‘x’ ‘X’

‘\”’ ‘\\’ ‘\n’

‘\0’ ‘\7’ ‘\33’

Les types de base
Types simples (4)

Constante_int



123 -12L

0377 0377L

0xFF 0XAB

Les types de base
Types simples (5)

```
#include <stdio.h>

int main ()
{
        unsigned int x;
        int y;
        y=-13;
        x=-12;
        printf (“\n %u” , x);
        printf (“ %d %d”, x,y);
}
```

Numération : Nombres réels (1)

Utilisation d'une arithmétique à virgule flottante

Forme exponentielle : SIGNE MANTISSE x Base $^{\text{Exposant}}$

En base décimale :

$5748{,}6954 = 57486954 \times 10^{-4}$

$= 0{,}57486954 \times 10^{4}$

$= 5{,}7486954 \times 10^{3}$

En binaire :

$1011{,}0111 = 1011\ 0111 \times 2^{-4}$

$= 0{,}1011011 \times 2^{4}$

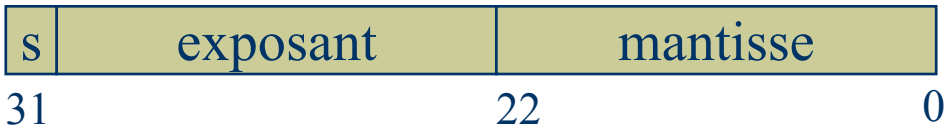
$= 1{,}0110111 \times 2^{3}$

Numération : Nombres réels (2)

Forme normalisée : $0{,}10110111 \times 2^{4}$

Pour utiliser au mieux la place disponible, la mantisse est normalisée de manière à éviter les zéros non significatifs, ce que l'on obtient en imposant $0{,}1 <= \text{mantisse} < 1$, ce qui garantit que le premier chiffre significatif est toujours celui situé immédiatement après le point décimal.

Représentation en machine sur 32 bits :



- Bit de signe : 0 pour + et 1 pour –
- Exposant sur 8 bits, des numéros 23 à 30, de –128 à +127
- Mantisse sur 23 bits, des numéros 0 à 22, normalisée (le premier bit est obligatoirement à 1, il est donc ignoré).

Numération : Nombres réels (3)

Exemple sur 32 bits :

Soit le nombre $37{,}625_{10}$ Forme binaire : 100101,1010

Normalisation : $0{,}100101101 \times 2^{6}$

Décomposition : signe : 0
exposant : 6
mantisse : 100101101

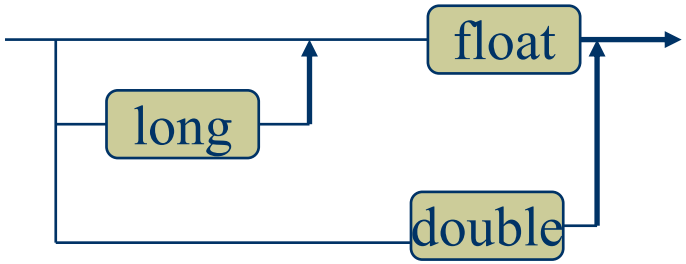
Exposant : $6 + 128 = 134_{10} = 10000110_2$ (afin d'éviter l'utilisation d'un bit de signe)

0	10000110	0010110100….

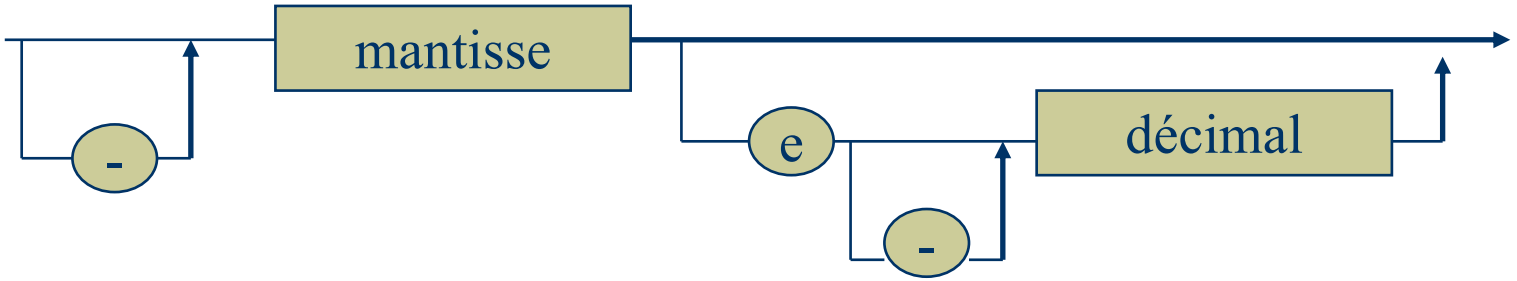
Nb max : 2^{127} pour exposant et mantisse proche de 1 d'où $2^{127} \sim 10^{38}$

Les types de base
Types simples (6)

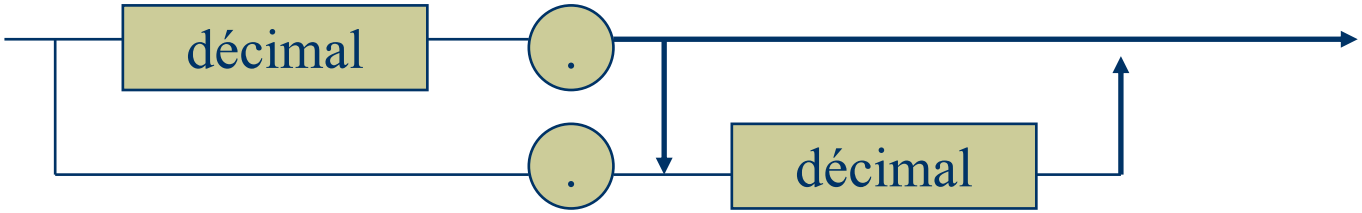
Type_flottant



Constante_flottante



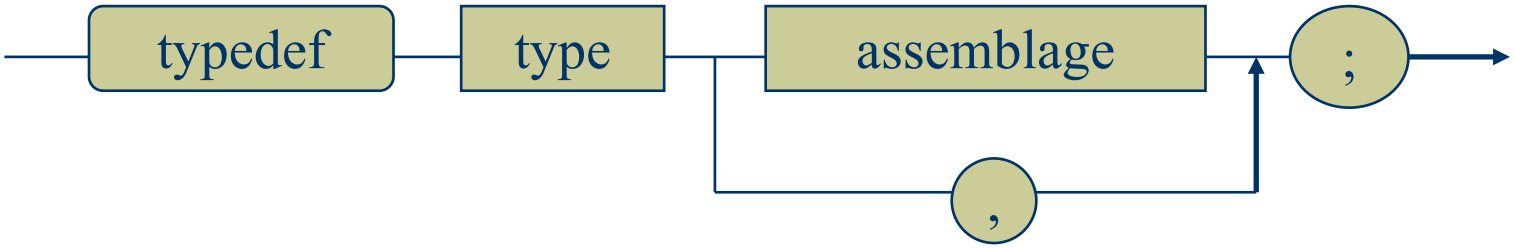
mantisse



-3.5e-6
.5
5.
3.e7

Définition de types

Définition de types



Assemblage (extraits)