3. Variables

- Toutes les variables doivent être déclarées explicitement avant leur première utilisation (on conseille de les déclarer toutes ensemble au début de la fonction).
- La déclaration n'initialise pas les variables à une valeur specifique.
- L'opérateur d'affectation (=) donne une valeur à une variable.
- Chaque variable est associée à une adresse dans la mémoire où se trouve "une petite boîte" de la taille correcte réservée pour stocker sa valeur.

On utilisera trois types de variables de base :

int pour des valeurs entières (*integer*). Taille (ordinateurs du magistère) : 4 octets (32 bits). Représentables (de façon exacte) : toutes les valeurs entières entre -2 147 483 648 et +2 147 483 647 ($= 2^{31} - 1$).

double pour des valeurs réelles (dites de double précision). Taille (ordinateurs du magistère) : 8 octets (64 bits). Représentables (de façon approximative avec une précision de 15 chiffres significatifs) : toutes les valeurs réelles positives et négatives avec valeur absolue entre $\approx 2.225 \cdot 10^{-308}$ et $\approx 1.798 \cdot 10^{+308}$.

char pour des caractères (*character*). Taille : 1 octet (8 bits). Ex. : `char c; c = 'k';` Remarque bien les guillemets simples pour un caractère, tandis que les chaînes de caractères s'écrivent avec des guillemets doubles.

Exemples :

```
int i = -100, bla = 0, compteur = 5;
double a = -0.5, bon_resultat = 2., c = -7.3e5, dx2 = 5.e-30;
const double G = 6.67e-11, kB = 1.38e-23;
```

- Selon la convention anglaise, on utilise un point et non pas une virgule comme séparateur décimal.
- Avec le point dans "2." (= 2.0) on dit au compilateur que le 2 doit être considéré comme un réel et non pas comme un entier.
- Notation scientifique : le 'e' veut dire "fois 10 puissance". Dans `5.e-30` le point ne veut pas dire "fois", mais c'est le séparateur décimale : `5.e-30` = `5.0e-30`. Note l'absence de parenthèses autour de l'exposant négatif.
- Avec le mot clé `const` on définit une constante au lieu d'une variable, dont on ne peut plus changer la valeur après.

Opérateurs arithmétiques

+ (addition), - (soustraction), * (multiplication), / (division)
Précédence : d'abord *, /, %, puis +, -. Utiliser des parenthèses pour changer.

Important : la division de deux entiers donne le quotient (partie entière) ; on trouve le reste avec l'opérateur % : 8 / 5 = 1, 8 % 5 = 3
Pour éviter une division euclidienne :

- Pour une valeur fixe rajouter un point pour convertir en réel : 8. / 5 = 1.6
- Dans le cas d'une variable faire un changement ponctuel de type :
  `int i = 8, j = 5; double a = (double)i / j;`

Exemples :

```
double resultat, a = 5., b = 0.5; resultat = (1 + a) * b / 2.;
int i = 2, j; j = 5 * i; j += 4; j++;
```

Commande d'affectation raccourcie : `j += 4;` est équivalent à `j = j + 4;`
De la même façon il existe aussi `-=`, `*=`, `/=`, `%=`.
Encore plus raccourci : `j++;` est équivalent à `j = j + 1;` (il existe aussi `--`).

Fonctions mathématiques

Pour utiliser les fonctions mathématiques il faut la bibliothèque math.h : mettre `#include<math.h>` au début du programme.

Puis on a accès à des fonctions comme `exp`, `log` (logarithme naturel), `log10` (logarithme base 10), `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`, `sqrt` (*square root* = racine carrée), `fabs` (valeur absolue) et des constantes comme `M_PI` (π) et `M_E` (e). L'argument d'une fonction se trouve obligatoirement entre parenthèses, par exemple `sqrt(2)`.

Il y a aussi la fonction `pow` (*power*) pour calculer une puissance : `pow(2.5, 1./3.)` donne la racine cubique de 2.5. Ne pas utiliser `pow` au lieu de la notation scientifique avec 'e' pour des puissances de 10 ! Un carré s'écrit souvent plus facilement comme une multiplication : `a*a` au lieu de `pow(a,2)`.

La fonction `ceil(x)` (*ceiling*) donne la valeur entière la plus petite mais $\geq x$, tandis que `floor(x)` donne la valeur entière la plus grande mais $\leq x$.

Les résultats de `ceil` et `floor` sont encore des réels, par exemple `floor(-3.4)` donne `-4.0`.