Informatique

Sequence 1 : Premiers pas

TD 1.1

v2025-09-11 P

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

Hello World

Sommaire

La division en langage C

En langage C, l'opérateur division ne se comporte pas de la même façon selon les types de données.

Si les deux variables sont des entiers : La division est une division euclidienne (division entière). Le résultat de l'opération est donc un ENTIER qui ne contient que le quotient de la division.

- Exemple : `5/2` retournera la valeur `2`

Si une des deux variables est flotante (float ou double) : La division est une division à virgule et le résultat sera donc un nombre à virgule.

- Exemple : `5/2.0` retournera la valeur `2.5`

L'opérateur modulo % – Le reste de la division euclidienne

L'opérateur modulo % retourne le reste de la division euclidienne.
Par exemple : `5%2` donnera 1, car 5 = 2*2 + 1.

1 Niveau 1

1.1 Variables

Exercice 1 : Informations personnelles

On souhaite stocker des informations simples concernant un étudiant :

- Son prénom
- Son âge
- Sa taille (en mètres)

Question 1 Pour chaque variable, proposer un type de données adapté et justifier le choix.

Question 2 Écrire les lignes de **déclaration** de ces variables (sans initialisation).

Question 3 Ecrire des lignes **d'initialisation** cohérentes pour un étudiant fictif.

Correction 1 : Informations personnelles

```
string prenom;
int age;
float taille;
prenom = "John";
age = 20;
taille = 1.74;
```

Exercice 2 : Affectations successives

On considère le programme suivant :

```
int x = 2;
int y = 5;
int z;
z = x + y;
y = z - 1;
x = y + z + x;
```

Question 1 Donner les valeurs finales de `x`, `y` et `z`.

Correction 2 : Affectations successives

```
// Etapes de calcul :
int x = 2;      // x = 2
int y = 5;      // y = 5
int z;          // z = ?
z = x + y;      // z = 2 + 5 = 7
y = z - 1;      // y = 7 - 1 = 6
x = y + z + x;  // x = 6 + 7 + 2 = 15

// Valeurs finales :
x = 15;
y = 6;
z = 7;
```

Exercice 3 : Types adaptés

On considère les informations suivantes :

- Nombre d'étudiants d'une promo
- Température extérieure
- Numéro de salle
- Prix d'un sandwich

Question 1 Pour chacune, donner le type le plus approprié et la déclaration de la variable.

Correction 3 : Types adaptés

```
// Nombre d'etudiants d'une promo : entier naturel
int nb_etudiants;

// Temperature exterieure : reel
float temperature;

// Numero de salle : souvent alphanumerique -> texte
string numero_salle;

// Prix d'un sandwich : reel en euros, préférer le double pour la précision (cf Niveau 3)
// (Remarque : on peut aussi stocker en centimes avec un int)
double prix_sandwich;
```

Exercice 4 : Prédire l'affichage (entier vs réel)

On exécute le programme suivant.

```
int a = 7, b = 2;
float x = 7, y = 2;
printf("A:%d\n", a / b);
printf("B:%f\n", x / y);
printf("C:%f\n", a / (float)b);
printf("D:%d\n", a % b);
```

Question 1 Si ce n'est pas déjà fait, lire l'encart sur la division, page 1

Question 2 Prédire l'affichage généré par les lignes ci-dessus.

Correction 4 : Prédire l'affichage (entier vs réel)

- a / b est une division entière (entiers) : $7/2 = 3$
- x / y est une division réelle (flottants) : $7.0/2.0 = 3.5$
- a / (float)b force une division réelle (cast) : 3.5
- a % b est le reste de la division euclidienne : $7 \% 2 = 1$

Affichage attendu :

```
A:3
B:3.500000
C:3.500000
D:1
```

1.2 Conditions

Exercice 5 : Majorité (partie 1)

On veut afficher « Majeur » si `age >= 18`, sinon « Mineur ».

Question 1 Décrire la logique en français ou pseudo-code avec un `if/else`. Donner deux exemples d'entrées et la sortie attendue.

Correction 5 : Majorité (partie 1)

```
// Pseudo-code :
// si age >= 18 alors afficher "Majeur"
// sinon afficher "Mineur"

// Traduction C (extrait) :
if (age >= 18)
{
    printf("Majeur\n");
}
else
{
    printf("Mineur\n");
}

// Exemples :
// age = 17 -> "Mineur"
// age = 21 -> "Majeur"
```

Exercice 6 : Qui est le plus grand ?

On souhaite écrire un programme qui affiche lequel de deux nombres `a` et `b` est le plus grand. S'ils sont égaux, le programme le dira.

Question 1 Proposer les lignes de codes qui, pour deux entiers `a` et `b`, affichent le plus grand ou s'ils sont égaux.

Correction 6 : Qui est le plus grand ?

```
if (a > b)
{
    printf("a est plus grand (%d)\n", a);
}
else if (b > a)
{
    printf("b est plus grand (%d)\n", b);
}
else
{
    printf("a et b sont egaux (%d)\n", a);
}
```

Exercice 7 : Nombre mystère (logique)

On considère un programme dont deux variables ont été déclarées : `nombre_mystere` et `nombre_joueur`. Le nombre mystère n'est pas connu du joueur. Il doit le deviner en tentant des valeurs au hasard.

```
int nombre_mystere = 7;
```

```
int nombre_joueur = get_int("Deviner le nombre mystere : ");
```

On veut écrire **bravo** si l'utilisateur à trouvé le nombre, **Raté** sinon.

Question 1 Ecrire un pseudo-code décrivant la logique à implémenter

Question 2 Traduire ces lignes en langage C

Question 3 Réécrire le programme pour afficher si le nombre proposé est **Trop petit** et **Trop grand**.

Correction 7 : Nombre mystère (logique)

```c
// 1) Pseudo-code :
// si nombre_joueur == nombre_mystere afficher "Bravo"
// sinon afficher "Rate"

// 2) Traduction C (version simple) :
if (nombre_joueur == nombre_mystere)
{
    printf("Bravo\n");
}
else
{
    printf("Rate\n");
}

// 3) Variante avec "Trop petit" / "Trop grand" :
if (nombre_joueur == nombre_mystere)
{
    printf("Bravo\n");
}
else if (nombre_joueur < nombre_mystere)
{
    printf("Trop petit\n");
}
else
{
    printf("Trop grand\n");
}
```

2 Niveau 2

2.1 Opérations et conversions

Exercice 8 : Opérations et priorités

Soit le code suivant :

```
int r = 10 - 2 * 3 + 8 / 2;
int s = r + 5 * (3 - 1);
```

Question 1 Donner l'ordre exact des opérations.

Question 2 Donner les valeurs finales de `r` et `s`

Correction 8 : Opérations et priorités

Ordre : * et / de gauche à droite, puis + et -. r : $10-(2\cdot 3)+(8/2) \to 10-6+4 \to 4+4 \to \boxed{8}$. s : $(3-1)=2$, $5\cdot 2=10$, donc $s=r+10=8+10=\boxed{18}$.

Exercice 9 : Conversions et cast

On considère :

```
int a = 13, b = 5;
float u = 13, v = 5;
int A = a/b;
float C = (float)a / b;
int D = (float)a / b;
int E = a%b;
```

Question 1 Donner les valeurs des variables A, B, C, D et E.

Le transtypage (cast)

En langage C, on peut momentanément considérer qu'une variable est d'un certain type au lieu de celui déclaré. Il suffit de spécifier le type entre parenthèses devant la variable.
Par exemple :

```
int variable_entiere;
float resultat = (float) variable_entiere/2;
```

Durant cette opération, `variable_entiere` sera considérée comme `float`.

Correction 9 : Conversions et cast

$A=\boxed{2}$ (division entière 13/5). $C=\boxed{2{,}6}$ (division réelle). $D=\boxed{2}$ (résultat réel 2,6 converti en `int` $\Rightarrow$ troncature). $E=\boxed{3}$ (reste de 13%5). *Remarque : B* n'est pas déclaré dans le code. S'il s'agit de u/v, alors $B=\boxed{2{,}6}$.

Exercice 10 : Prédire l'affichage (mélange de types)

```
int n = 9, d = 4;
float p = 9, q = 4;
n = n - d/2;
p = p / q;
printf("A:%d\n", n);
printf("B:%f\n", p);
printf("C:%f\n", (float)n / d);
printf("D:%d\n", n % d);
```

Question 1 Prédire chaque ligne affichée et expliquer le rôle de la **division entière** dans la mise à jour de `n`.

Correction 10 : Prédire l'affichage (mélange de types)

$d/2$ est une **division entière** $(4/2 = 2)$, donc $n = 9 - 2 = \boxed{7}$.
p/q est réel $(9/4 = \boxed{2{,}25})$.
(float)$n/d = 7{,}0/4 = \boxed{1{,}75}$. $n\%d = 7\%4 = \boxed{3}$.
Affichage : `A:7`, `B:2.250000`, `C:1.750000`, `D:3`.

2.2 Conditions multiples et booléens

Exercice 11 : Intervalle inclusif et exclusif

Question 1 Ecrire un condition en langage C pour tester $5 \leq x \leq 10$.

Question 2 Ecrire ce même test d'une façon différente.

Question 3 Expliquer pourquoi `(x > 5 || x < 10)` est incorrecte.

Question 4 Expliquer pourquoi `(5 <= x <= 10)` est incorrecte.

Correction 11 : Intervalle inclusif et exclusif

Test direct : `(x >= 5 && x <= 10)`.
Variante équivalente : `!(x < 5 || x > 10)`.
`(x > 5 || x < 10)` est toujours vraie (union des deux demi-axes).
`(5 <= x <= 10)` évalue `(5 <= x)` en 0/1, puis compare à 10 ⇒ toujours vrai.

Exercice 12 : Autorisation d'accès

On considère trois variables booléennes (le type **bool** est défini dans la bibliothèque `stdbool`) :

```
bool isRegistered;
bool hasBadge;
bool isAdmin;
```

On veut accorder l'accès à une personne si elle possède un badge et est enregistrée. L'accès sera aussi accordé si la personne est administratrice.

Question 1 Donner une condition booléenne correctement parenthésée qui accorde, ou non, l'accès.

Correction 12 : Autorisation d'accès

Condition : `( (hasBadge && isRegistered) || isAdmin )`.

Exercice 13 : Calcul d'expressions booléennes

Soient `a=3, b=7, c=7`.

Question 1 Évaluer les expressions booléennes suivante (pour chacune, dire si elle renvoie **vrai** ou **faux**) :

- `(a<b) && (b==c)`
- `!(a<b) && (b==c)`
- `(a<b) || (b==c)`
- `(a>=b) || (c!=7)`
- `(a<b) && !(b==c)`
- `!(a<b) || (b==c)`

Correction 13 : Calcul d'expressions booléennes

Avec $a = 3$, $b = 7$, $c = 7$:
$(a < b) \wedge (b == c)$: **vrai**.
$(a \geq b) \vee (c \neq 7)$: **faux**.
$\neg(a < b) \wedge (b == c)$: **faux**.
$(a < b) \wedge \neg(b == c)$: **faux**.
$(a < b) \vee (b == c)$: **vrai**.
$\neg(a < b) \vee (b == c)$: **vrai**.

3 Niveau 3

Les nombres flottants en C

En langage C, il existe deux types principaux pour représenter des nombres à virgule :

float : type virgule flottante simple précision (32 bits). Il permet de représenter environ 7 chiffres significatifs avec une plage de valeurs allant de 10^{-38} à 10^{38}. C'est rapide et économique en mémoire, mais moins précis.

double : type virgule flottante double précision (64 bits). Il permet de représenter environ 15 à 16 chiffres significatifs avec une plage beaucoup plus large (10^{-308} à 10^{308}). Il est plus précis mais consomme deux fois plus de mémoire.

Attention : les nombres flottants ne sont qu'une approximation des réels. Certaines opérations ne donnent pas toujours le résultat attendu exactement, à cause des limites de la représentation binaire.

- Exemple avec `float` : `1.0f/3` donnera une valeur approchée de 0.333333.
- Exemple avec `double` : `1.0/3` donnera une valeur plus précise, par exemple 0.3333333333333333.
- Exemple de limite : la somme de `0.1+0.1+0.1` peut afficher 0.30000000000000004.

3.1 Traces, erreurs et structures

Exercice 14 : Exécution pas à pas

Analyser l'exécution suivante pas à pas :

```
int a = 5, b = 10, c = 0;
c = a + b;      // 1
b = c - a;      // 2
a = a + 1;      // 3
c = a + b + c; // 4
```

Question 1 Donner les valeurs finales de `a`, `b` et `c`.

Correction 14 : Exécution pas à pas

Étapes : $c = 15$, $b = 10$, $a = 6$, puis $c = 6 + 10 + 15 = 31$. **Final :** $a = 6$, $b = 10$, $c = 31$.

Exercice 15 : Précision et débordements

Question 1 Expliquer ce qui peut arriver pour `int x = 2000000000 + 2000000000;`. Donner un exemple d'impact réel.

Question 2 Comparer `float` vs `double` pour accumuler des montants (somme de 1/10 répété 100 fois).

Correction 15 : Précision et débordements

Débordement : la somme dépasse `INT_MAX` (32 bits). En C, le dépassement d'entier *signé* est *indéfini* (comportements possibles : wrap-around, valeur erronée, optimisations trompeuses). Impact : compteurs, minuteries, totaux financiers qui « bouclent ».
Exemple : La fusée arianne 5 s'est autodétruite en 1996 à cause d'un dépassement d'entier dans le calcul de la trajectoire.
Accumulation : additionner 0.1 100 fois n'aboutit pas exactement à 10. En `float`, l'erreur relative est notable (affiche souvent 9.999999 ou 10.000001) ; en `double`, l'erreur est bien plus faible. Pour des montants, préférer `double` (voire des entiers en centimes).

Exercice 16 : Prédire l'affichage (mélange et cast)

```c
int i = 7, j = 3;
float u = 7, v = 3;
i = i + j/2;        // division entière
u = u / v;          // division réelle
printf("A:%d\n", i);
printf("B:%f\n", u);
printf("C:%f\n", (float)i / j);
printf("D:%d\n", (i * j) % 5);
```

Question 1 Prédire chaque ligne d'affichage et expliquer l'effet du cast et de la division entière sur `i`.

Correction 16 : Prédire l'affichage (mélange et cast)

A : $j/2$ est entier $(3/2 = 1)$, donc $i = 7 + 1 = 8 \Rightarrow$ `A:8`. **B :** u/v réel $= 7/3 \approx 2{,}333333 \Rightarrow$ `B:2.333333`. **C :** (float)$i/j = 8/3 \approx 2{,}666667 \Rightarrow$ `C:2.666667`. **D :** $(i \cdot j)\%5 = (8 \cdot 3)\%5 = 24\%5 = 4 \Rightarrow$ `D:4`. Effet : la division entière réduit $j/2$ à 1 (pas de décimales) ; le *cast* force une division réelle.

3.2 Imbrications et logique avancée

Exercice 17 : Écrire ses propres conditions

On dispose de drapeaux `badgeOK`, `tenueOK`, `formationOK`, `isResponsable`.
Règles d'accès :

- Les responsables sont autorisés quelque soit leur tenue, l'état de leur badge ou de leur formation.
- Les autres seront autorisés s'ils sont en possession d'un badge et soit qu'ils ont passé une formation, soit qu'ils possèdent une tenue de sécurité

Question 1 Écrire l'expression booléenne correspondante.

Correction 17 : Écrire ses propres conditions

```c
if ( isResponsable || (badgeOK && (formationOK || tenueOK)) ) {
    /* accès accordé */
} else {
    /* accès refusé */
}
```

Exercice 18 : Majorité avancée (partie 3)

Une billetterie donne un tarif réduit si l'âge de la personne est un multiple de 10. Le tarif sera majoré pour les plus de 60 ans.

Question 1 Proposer une structure `if/else if/else` lisible.

Correction 18 : Majorité avancée (partie 3)

Hypothèse claire : la majoration > 60 **prime** sur la réduction « multiple de 10 ».

```c
if (age > 60) {
    printf("Tarif majore\n");
} else if (age % 10 == 0) {
    printf("Tarif reduit\n");
} else {
    printf("Tarif normal\n");
}
```

Exercice 19 : Court-circuit et sécurité

Analyser les deux versions suivantes :

```c
if ( i / j > 2 && j != 0 ) {
    printf("OK\n");
}
```

```c
if ( (j != 0) && (i / j > 2) ) {
    printf("OK\n");
}
```

Question 1 Quelle différence y a-t-il entre ces deux versions ?

Question 2 Laquelle vous semble la plus efficace et pourquoi ? Justifier votre réponse et donner un exemple.

Correction 19 : Court-circuit et sécurité

Avec l'opérateur `&&`, l'évaluation est **court-circuitée** de gauche à droite.
À gauche la division : la première version peut diviser par zéro si `j==0` (dangereux/UB).
À gauche le test de zéro : la seconde est **sûre** : si `j==0`, la division n'est pas évaluée.
Efficacité : la seconde évite une division inutile quand `j==0` (et protège contre l'erreur).
Exemple : `i=10, j=0` — 1re version : erreur potentielle ; 2e : pas de division.