Langage et Programmation – 3ème séance

Fabien Tarissan

1 Premiers pas

Exercice 1 — *Opérations simples* Écrire les expressions permettant de :
- calculer le résultat d'un expression arithmétique (par exemple $\frac{3+9}{2*3}$)
- calculer le quotient de la division euclidienne de 42 par 5
- calculer le reste de la division euclidienne de 42 par 5
- calculer le résultat de la division réelle de 42 par 5
- tester l'égalité entre 6 et $2*3$.
- concatener la chaîne "j'enseigne " avec la chaîne "la programmation dite "impérative""

Exercice 2 — *Interagir avec l'utilisateur*
- Écrire un programme qui affiche à l'écran "Bonjour !"
- Écrire un programme qui demande à l'utilisateur de rentrer son nom, le mémorise dans une variable et affiche à l'écran "Bonjour" suivi du nom contenu dans la variable
- Écrire un programme qui demande à l'utilisateur de rentrer son année de naissance et affiche à l'écran son age.

2 Fonctions simples

Exercice 3 — *Fonction mystère* Soit le code suivant :

```
double mystere (double a, double b, double c) {
  if (a<b) {
     if (a<c) return a;
     else return c;
  } else {
     if (c<b) return c;
     else return b;
  }
}
```

1. Quel est le résultat de l'application de `mystere(1.0,3.5,5.9)` ?
2. Que fait la fonction ? Renommer la fonction et lui donner une entête correcte.
3. Combien de tests différents faut-il faire pour s'assurer que la fonction est correcte ?

Exercice mettant en avant l'importance de bien spécifier ce que fait la fonction : donner un nom qui indique ce qu'elle fait et bien préparer l'entête de la fonction.
Importance également de proposer un jeu de test complet.

De façon générale, les exercices de type « fonctions mystères » sont utiles pour tester la compréhension que les élèves ont des programmes. Permet aussi de demander l'évaluation pas-à-pas d'une application de fonction.

Exercice 4 — *Fonction de conversion*
- Écrire une fonction `farhenheitEnCelsius` qui convertit une température donnée en farhenheit en celsius ($c = \frac{5*(f-32)}{9}$. Refléchir aux types des arguments et de la valeur de retour.

– Proposer une fonction qui réalise la conversion inverse de la précédente

Exercice qui met en évidence ce qui fait partie de la programmation et ce qui fait partie de l'algorithmique. Permet de clairement découper les 4 étapes nécessaires pour écrire une fonction :

1. *Écrire l'entête de la fonction*
2. *Trouver l'algorithme pour résoudre le problème (très simple pour la première partie de cet exercice puisque la solution est donnée explicitement ; il faut simplement résoudre l'équation en f pour la seconde partie)*
3. *Traduire l'algorithme dans le langage*
4. *Tester la fonction*

Exercice 5 — *Entier naturel ?* Écrire une fonction `naturel` qui teste si un entier est un entier naturel ou non.

Introduit simplement le IF. Plein de variantes possibles. Par exemple calcul de la valeur absolue d'un nombre.

Exercice 6 — *Divisible ?* Écrire une fonction `divisible` qui, étant donnés deux nombres entiers, teste si le premier est divisible par le second.

Manipulation des opérateurs % et / pour la division euclidienne.

Exercice 7 — *Nombre premier* À l'aide de la fonction précédente, écrire une fonction `nbPremier` permettant de déterminer si un nombre donné est *premier* ou non.

Quelques exercices complémentaires simples à partir de cette fonction :
– Afficher tous les nombres premiers inférieurs à un entier n (en paramètre)
– Afficher les n premiers nombres premiers (n étant un paramètre)

Exercice 8 — *Trouver l'erreur* Dans ce qui suit, il est demandé de prédire ce que vont faire les programmes avant de les tester.

– Soit le code suivant :

```
boolean malade(double x) {
  if (x=37.5) return true;
  else return false;
}
```

Que va-t-il se passer lors de l'appel `malade(39.0)` ? Corriger le code.

Mise en évidence de la différence entre les opérateurs = et ==.

– Soit le code suivant :

```
double x= 4/3;
println(x);
```

Qu'affiche ce bout de programme ? Pourquoi ?

Mise en évidence du rôle de l'opérateur / et de l'ordre dans lequel les calculs sont faits. On évalue en premier l'expression 4/3 !
Permet de rappeler également que l'instruction T var = expr ; *est en fait une double instruction pour le langage :* T var ; *puis* var = expr ;.

– Soit le code suivant :

```
if (3<5) x=6/2;
else x=6/0;
```

Que fait le programme lors de l'exécution de ce code ?

Mise en évidence de la sémantique du IF qui n'exécute qu'une seule des deux branches

.

– Soit maintenant la fonction :

```
int si(boolean b, int e1, int e2) {
  if (b) return e1;
  else return e2;
}
```

Que se passe-t-il lors de l'appel de fonction `si(3<5, 6/2, 6/0);` ?

*Mise en évidence du fonctionnement d'un appel de fonction : **tous** les arguments dans l'appel sont évalués !*

De façon générale, les exercices de type « trouver l'erreur » sont très utiles pour tester la compréhension d'un langage de programmation sans s'embarrasser de la programmation effective des fonctions demandées. C'est un bon outil pédagogique qui exige d'avoir bien assimiler la sémantique des instructions considérées.

Exercice 9 — *Échange de valeurs* Soit la fonction suivante :

```
void echange(int x, int y) {
  int tmp=x;
  y=x;
  x=tmp;
}
```

Que vaut la variable `a` après l'execution suivante :

```
int a = 1;
int b = 5;
echange(a,b);
}
```

Commenter le résutat

Mise en évidence du mécanisme de passage par valeur.

3 Les tableaux

Exercice 10 — *Premières opérations*

- Créer un tableau `t` contenant les valeurs | 1 | 3 | 5 |.
- Afficher la taille de `t`
- Écrire une fonction `afficheTableauInt` permettant l'affichage d'un tableau d'entiers passé en argument.
- Écrire un programme demandant à l'utilisateur un entier n et créant un tableau `tab0n` contenant les valeurs de 0 à n (inclus).
- Écrire un programme demandant à l'utilisateur un entier n, créant un tableau de taille n et demandant ensuite à l'utilisateur les valeurs à mémoriser dans le tableau. [1]
- Écrire une fonction qui, étant donnés un tableau d'entiers, et deux indices `i` et `j`, échange les valeurs de ces deux indices si c'est possible, ne fait rien sinon (*i.e.* si les indices ne correspondent pas à des cases du tableau).

Exercice 11 — *Moyenne* Écrire une fonction `moyenne` qui calcule la moyenne des éléments d'un tableau d'entiers.

Variante plus simple : calculer la somme des éléments d'un tableau.
Exercice complémentaire : calculer l'écart-type.

1. il sera utile d'encapsuler primitive dans une fonction `int [] readTableauInt(int n)` afin de la réutiliser par la suite.

Exercice 12 — *Min et max* Soit un tableau d'entiers naturels non vide. Écrire une fonction `maxTableau` qui calcule le maximum contenu dans le tableau. Donner la variante calculant le minimum.

En complément, calculer l'écart maximum dans un tableau.

Exercice 13 — *Appartenance* Écrire une fonction `appartient` qui teste l'appartenance d'un entier dans un tableau.

Exercice 14 — *Occurence* Écrire une fonction `nbOccurence` qui compte le nombre d'occurence d'un entier dans un tableau.

En complément (mais plus dur) : calculer la distribution des valeurs du tableau, c'est-à-dire, pour chaque valeur v du tableau, le nombre d'occurence de v dans le tableau.

Exercice 15 — *Occurence du maximum* Écrire une fonction `nbOccurenceMax` qui compte le nombre d'occurence du maximum dans un tableau. Quelle est la complexité de votre fonction. Peut-on faire mieux ?

Exercice 16 — *Égalité* Qu'affiche le code suivant :

```
int [] t1={1,2,3};
int [] t2={1,2,3};
println(t1==t2);
```

Expliquer pourquoi. Proposer une fonction qui teste l'égalité de deux tableaux d'entiers.

Mise en évidence de la sémantique de l'opérateur ==. C'est l'égalité dite physique.

Exercice 17 — *Même contenu* Écrire une fonction qui teste si deux tableaux d'entiers de même taille contiennent les mêmes valeurs. On supposera dans un premier temps que tous les éléments d'un tableau sont distincts. Que pensez-vous de la complexité de votre fonction. Peut-on faire mieux ?

Refléchir à la variante dans laquelle les éléments peuvent être répétés.

La complexité est en $O(n^2)$. On peut faire mieux si on connaît les algorithmes de tris ($O(nlog(n))$), ce qui permet également de répondre efficacement à la variante autorisant la duplication des éléments.

Exercice 18 — *Tri* Implémenter les algorithmes de tris vus en cours d'algorithmique.

4 Les fonctions récursives

Exercice 19 — *Somme des n premiers entiers* Écrire une fonction récursive `sommeN` qui, étant donné un entier n, renvoie la valeur de $\sum_{i=1}^{n} i$.

En complément. On peut demander aux élèves de vérifier que cette somme est bien toujours égale à $\frac{n(n-1)}{2}$. La notion de toujours *ici permet également d'aborder la notion de preuve puisque ce genre de calcul bien entendu ne* ***prouve pas*** *l'égalité. Pour le faire avec un ordinateur, il faut utiliser d'autres langages de programmtion, dédiés à la preuve de propriétés.*
Variantes possibles calculant la somme des n premiers entiers pairs (resp. impairs).

Exercice 20 — *Calcul du* PGCD On rappelle que l'algorithme d'euclide permet de déterminer le PGCD de deux entiers à l'aide des divisions euclidiennes en remarquant que :
- $\text{pgcd}(n_1, n_2) = n_1$ si $n_2 = 0$.
- $\text{pgcd}(n_1, n_2) = pgcd(n_2, r)$ sinon où r est le reste de la division euclidienne de n_1 par n_2 (on suppose $n_2 \leqslant n_1$).

À partir de ce principe, écrire une fonction récursive `pgcd` calculant le PGCD de deux entiers. En déduire une fonction `premiersEntreEux` testant si deux entiers sont premiers entre eux.

Variantes possibles avec d'autres définition de la fonction pgcd. Par exemple :
- $\text{pgcd}(n_1, 0) = n_1$
- $\text{pgcd}(0, n_2) = n_2$
- $\text{pgcd}(n_1, n_2) = pgcd(n_1, n_2 - n_1)$ *si* $n_1 < n_2$
- $\text{pgcd}(n_1, n_2) = pgcd(n_1 - n_2, n_2)$ *sinon*

Exercice 21 — *Puissance* Écrire une fonction récursive `puissance` qui, étant donnés un nombre x et un entier n, renvoie la valeur de x^n.
Écrire également sa variante itérative.
Combien d'appels de fonction (ou de passages dans la boucle) sont faits lors de l'appel à `puissance(x,n)` ?

Exercice 22 — *Puissance efficace* En s'appuyant sur la définition ci-dessous, proposer une fonction récursive `puissanceEfficace` plus efficace que la précédente pour calculer x^n :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ (x^{n/2})^2 & \text{si } n \text{ est pair} \\ x * (x^{n/2})^2 & \text{si } n \text{ est impair} \end{cases}$$

Combien d'appels de fonction sont faits lors de l'appel à `puissanceEfficace(x,n)` ?

Les exercices 21 et 22 permettent de mettre en évidence qu'il n'y a pas que le résultat qui compte. Le temps mis par la machine à répondre est également un critère important (lien avec l'algorithmique et la complexité).

Exercice 23 — *Approximation de la racine carrée* Le but de cet exercice est d'écrire une fonction qui calcule une valeur approchée de la racine carrée d'un nombre. Soit x un nombre réel positif, une valeur approchée de $\sqrt{x}$ est donnée par le calcul des valeurs de la suite suivante :

$$u_n = \begin{cases} 1 & \text{si } n = 0 \\ \frac{u_{n-1} + \frac{x}{u_{n-1}}}{2} & \text{sinon} \end{cases}$$

Écrire une fonction qui, étant donnés un nombre x et un entier n, renvoie l'approximation au rang n de $\sqrt{x}$. Attention à l'efficacité de votre fonction [2]

Variante pour calculer une appxoximation de π en utilisant la relation suivante :

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \ldots + \frac{1}{n^2} + \ldots = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2}$$

Idem pour calculer $\cos(x)$ *:*

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots = \lim_{n\to\infty} \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{(2k)!}$$

De telles variantes existent pour le calcul du sinus, de e, ...

Exercice 24 — *Fibonacci* Écrire une fonction récursive `fibonacci` qui, étant donné un entier n, renvoie la valeur de la suite de Fibonacci au rang n :

$$\text{fib}_n = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ \text{fib}_{n-1} + \text{fib}_{n-2} & \text{sinon} \end{cases}$$

Donner la variante itérative. Que pensez-vous de la complexité des deux variantes ?

2. plein de variantes avec d'autre valeurs : approximation de π, de e, ...

5 Des exercices en vrac

Exercice 25 — *D'autres exercices* Sans liens les uns avec les autres mais groupés par intérêt :

1. Écrire une fonction qui calcule la distance parcourue en fonction de la vitesse et du temps.
2. Écrire une fonction qui calcule la période d'un pendule en fonction de sa longueur.
3. Écrire une fonction qui renvoie la valeur absolue d'un nombre réel.
4. Écrire une fonction qui calcule la valeur d'un polynôme du second degré en x (4 paramètres)
5. Écrire une fonction calcule les racines d'un polynômee du second degré (3 paramètres)

   Sur les entiers naturels :
6. Écrire une fonction qui donne la liste des diviseurs d'un entiers
7. Écrire une fonction qui décompose un nombre en facteur de nombres premiers
8. Écrire une fonction qui, étant donnés deux entiers n et m (avec $m \geqslant n$) crée un tableau contenant tous les nombres compris entre n et m (inclus).
9. Écrire une fonction qui affiche les n premiers nombres premiers
10. Écrire une fonction qui affiche les entiers naturels inférieurs à un entier n en utilisant le crible d'Eratosthène. Comparer son efficacité avec la solution qui découle de l'exercice 7.

    Sur les tableaux :
11. Écrire une fonction qui calcule l'écart-type d'un tableau de nombres.
12. Écrire une fonction qui calcule l'écart maximum dans tableau de nombres.
13. Écrire une fonction qui teste si un tableau de nombre est croissant.
14. Écrire une fonction qui calcule la distribution des valeurs d'un tableau de nombres.
15. Écrire une fonction qui inverse l'ordre des éléments d'un tableau (variante qui fait la modification *en place* (c'est à dire change les valeurs du tableau donné en paramètre) et variante qui crée un nouveau tableau.

    Sur la récursion :
16. Écrire une fonction qui calcule la somme des n premiers nombres pairs.
17. Écrire une fonction qui calcule une approximation de π, de $cos(x)$: cf exercice 23
18. Écrire une fonction qui calcule le coefficient binomial

$$C_p^n = \begin{cases} 1 & \text{si } p = 0 \\ 1 & \text{si } n = p \\ C_{p-1}^{n-1} + C_p^{n-1} & \text{sinon} \end{cases}$$