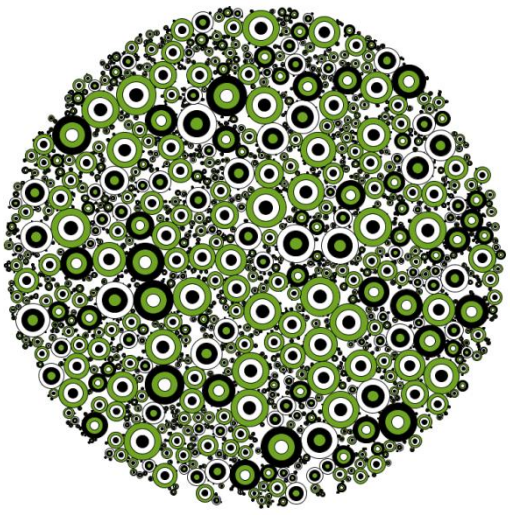
Algorithmique et structures de données

ESGI – 1i

Septembre 2021

Frédéric Baudoin

baudoin2020@gmail.com



Objectifs du cours

Compétences

Le cours vise à pouvoir

- **écrire** des programmes sur papier (sans machine)
- en mettant en œuvre une solution de type **force brute (** où toutes les solutions possibles sont examinées) en séparant
  - l'énumération des éléments de l'espace de recherche et
  - le traitement attendu
- les traitements concernés peuvent être
  - filtrer
  - compter
  - calculer le maximum des valeurs que prend une variable (ou le minimum)

Connaissances

Le cours passe en revue les thématiques classiques qui entrent en jeu dans la programmation élémentaire : **boucles, fonctions, tableaux, récursivité,**

Les bonnes pratiques de l'écriture des algorithmes (nommage, lisibilité, abstraction, …) seront abordées.

Qu'est ce qu'un algorithme ?

Un algorithme est (la description d') une procédure à suivre pour résoudre un problème : à partir d'un algorithme, on peut déterminer la séquence d'**instructions** conduisant à la résolution du problème.

Il existe plusieurs sortes **d'instructions** :

- les instructions d'entrée
- les instructions de sortie
- les instructions internes

Le cas de l'interactivité

Exemple : l'algorithme d'un distributeur de café

Algorithme

si la somme insérée est suffisante pour la boisson choisie,

préparer cette boisson et

rendre la monnaie

Exemple : les algorithmes pour les ordinateurs

Algorithme de l'écho

Mémoriser ce qui est saisi au clavier.

Puis l'afficher sur l'écran

- Instructions d'entrée : récupération d'une saisie au clavier, d'un clic de souris
- Instructions de sortie : affichage à l'écran, émission de sons
- Instructions internes : mémorisation

Mise en œuvre du mécanisme de mémorisation : la table des variables

Nom de la variable	Type	Valeur	Emplacement mémoire
X	Nombre entier	5	0007 1CF0 → 0008 A3BB
Y	Binaire	1	0007 1CF0 → 0007 1D34
…			

Chapitre 1

Expressions

Types de données

Variables

Instructions

Structures conditionnelles

A. Types, expressions, variables, instructions, algorithme

1. Types de données, variable

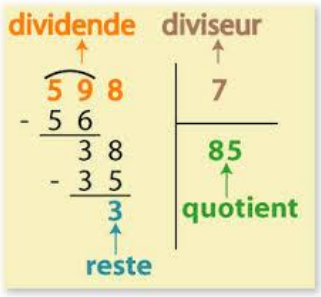
Le type entier

Pseudo code	Processing	C
Ent	int	

Ensemble de valeurs	Nombres entiers relatifs
Opérateurs	Arithmétiques : + ,- ,* , ^ Euclidiens : division entière / et modulo %
Opérateur de relation	= =, !=, <, <=, >, >=

L'opérateur de division entière « / » et le quotient de la division euclidienne

Permet de calculer **le quotient**. Par exemple, 598/7=85



L'opérateur modulo « % » et le reste de la division euclidienne

Si x et y sont des entiers positifs[1] x%y vaut **le reste** de la division euclidienne de x par y : 598%7=3 et on a :

$$x = y * (x/y) + x \,\%\, y$$

(598=85*7+3)

Questions :

- Suffit il d'avoir 22 = 5*3 + 7 pour écrire 22%5=7 ?
- Combien valent respectivement 46%7 , 7%46 , 7%1 , 1%7 , 0%7 et 7%0 ?

[1] Si on autorise les valeurs négatives pour x et y, il existe plusieurs possibilités :
définition 1 : $x \,\%\, y = reste\ de\ la\ division\ euclidienne\ de\ x\ par\ y$. Le diviseur et le modulo ne peuvent pas être négatifs. Si on autorise les valeurs négatives pour le diviseur, on doit avoir $0 \leq r < |y|$
définition 2 : $x \,\%\, y = x - y * Ent(x/y)$. Le modulo est toujours compris entre 0 (inclus) et le diviseur y (exclu) et qui a le même signe que le diviseur y :
définition 3 : $x \,\%\, y = x - y * troncatureDecimale(x/y)$. Le modulo a le même signe que le dividende. C'est cette définition qui prime en C, C++ et Java : $troncatureDecimale(-2{,}6) = -2$

Le type reel

Pseudo code	Processing	C
Reel	float	

Ensemble de valeurs	Nombres décimaux
Notation des valeurs littérales[2]	-3.0, 3.14, 1.12E-4
Opérateurs	+, - ,*, /, ^
Opérateur de relation	= =, !=, <, <=, >, >=

Le type booléen

Pseudo code	Processing	C
Bool	boolean	Ø
ET, OU, NON	&&, ||, !	

Notation du type	BOOL
Ensemble de valeurs littérales	VRAI FAUX
Opérateurs logiques	OU, ET , NON
Opérateur de relation	= =, !=

Table de vérité des opérateurs booléens :

X	**Y**	**X ET Y**	**X OU Y**	**NON X**
FAUX	FAUX	FAUX	FAUX	VRAI
VRAI	FAUX	FAUX	VRAI	FAUX
FAUX	VRAI	FAUX	VRAI	-
VRAI	VRAI	VRAI	VRAI	-

Le type caractère

Pseudo code	Processing	C
Car	char	

Ensemble de valeurs	Tout caractère ayant une représentation graphique (a-z, A-Z, 0-9, ponctuation)
Notation des valeurs littérales	Entre deux apostrophes. Par exemple 'u '
Opérateur de relation	= =, !=, <, <=, >, >= L'ordre est relatif à UNICODE. Par exemple, 'A '< 'B ' <…< 'a '< 'b '

Le type chaine de caractères

Pseudo code	Processing	C
Chaine	String	char[] ou char*

Ensemble de valeurs	Toute suite de caractères.
Notation des valeurs littérales	Entre deux guillemets. Par exemple, "chaine "
Opérateur de relation	= =, !=
Opérateur de concaténation	+. Par exemple "bon "+ "jour " vaut "bonjour "

[2] La notion de littéral en informatique renvoie à celle de constante en maths

2. Expressions

Les expressions sont formées de variables et/ou de valeurs littérales, liées par des opérateurs (et éventuellement des parenthèses).

Exemple d'expressions : 3+x, "toto"+y, (1+2)*4 …

EN ALGO, VARIABLES, VALEUR LITTERALES et OPERATEURS DOIVENT APPARTENIR AU MEME TYPE. A l'inverse des langages informatiques (C, Java), qui autorisent une certaine hétérogénéité dans les expressions. (voir conversion – ou cast - implicite ou explicite)

Par exemple,

- 1+2 est une expression bien formée car 1 et 2 sont des entiers, et + est un opérateur entier.
- 'e'+'r' n'est pas une expression bien formée car + n'est pas un opérateur du type caractère
- 'e'+2 n'est pas une expression bien formée car 'e' et 2 ne sont pas de même type.

Exercices

Exercice 0 Evaluation d'expressions
Donner le type de l'expression et précisez sa valeur :

Expressions	Type	Valeur
13+8		
3*(2-4)		
34/10		
34.0/10.0		
34%10		
10%34		
34/0		
5+3<4		
5-4>3 OU 3*4==9+3		
5+3>=7 ET NON (5-3==8)		
5+2>4 ET NON (VRAI OU 1==2)		
x=1+2		
'1'+2		
'b'>='B'		
"13"+"8"		

3. Variables

Cycle de vie des variables

1	Déclaration	On indique le type et le nom	ENT i
2	Initialisation	On affecte[3] une valeur	i = 2
3	Utilisation	On manipule la variable dans des expressions	j = i + 2

4. Instructions

4.1 L'instruction d'affectation (mémorisation)

Pseudo code	Processing	C
`var = expr`		

Fonctionnement :

1. on calcule la valeur de l'expression expr
2. on impose cette valeur à la variable *var*.

4.2 Les instruction d'entrée – sortie :

Instruction d'entrée : saisie clavier

Pseudo code	Processing	C
saisir(var) saisir(var1, var2, …)	Ø	scanf("%d", var) si var est de type int

Fonctionnement :

1. La main est donnée à l'utilisateur qui entre au clavier une valeur
2. cette valeur est affectée à la variable *var*

Instruction de sortie :affichage console

Pseudo code	Processing	C
afficher(expression) afficher(exp1, exp2, …)	print(expression) print(exp1, exp2, …)	printf("%d", expression) si expression est de type int

Fonctionnement : La valeur de l'expression est affichée à l'écran.
Remarque :

- Afficher("to"+ "ta") équivaut à Afficher("to", "ta")
- caractère de passage à la ligne : alaligne

[3] '=' désigne l'opérateur d'affectation usuel

4. **Algorithme**

Exercice 1 : Exemple d'algorithme 1 : calcul d'expressions

Donner l'évolution de la table des variables (hors emplacement mémoire) dans l'algorithme ci-dessous et dites ce qui est affiché sur la console (à l'écran). En cas d'erreurs, poursuivez quand même l'exécution du code.

```
ALGO exemple1
      ENT x,y=3,z=9
      BOOL rep
      Afficher(5)
      Afficher(rep)
      x=3
      x=2
      rep = VRAI
      Afficher(x)
      x=x+1
      x=x+1
      rep=4
      rep=  NON rep
      y=x-y
      x=x-y
      Afficher(x+2, x+y, x%z, rep)
      Afficher(x+2,"x+y", 'x%z', NON rep)
FIN exemple1
```

Console :

VARIABLE	TYPE	Evolution des valeurs
x		
y		
z		
rep		

Exercice 2 : Exemple d'algorithme 2 : instruction vs expression

Un algorithme est une suite d'instructions. Dites si l'algorithme suivant comporte des erreurs de syntaxe

```
ALGO exemple0
      ENT x,y,z
      x=2
      Afficher(x)
      y=x+y*
      Afficher(x=1+2)
      1+2
      x=Afficher(3+1)
      x+2=x
```

Exercice 2.5

Les lignes du code suivant ont étés mélangées. Retrouver l'ordre initial qui permet d'avoir sur la console la sortie désirée.

Il est interdit :

- de modifier le contenu des lignes
- d'utiliser une ligne plusieurs fois
- ne pas utiliser toutes les lignes

Algo mélangé	Console observée	Algo réarrangé	Console désirée

ALGO
 	
`ent x= 2`
`x=x+1`
`afficher (x)`
`x=x+1`
`x=x+1`
`x=x+1`
`x=x+1`
`x=x+1`
 	
FIN
			`5`

ALGO
 	
`ent x= 2, y=3`
`afficher (y)`
`x=3*y+3`
`x=y+1`
`x=y*y`
`x=y-x`
 	
FIN
			`5`

ALGO
 	
`z=x-y`
`afficher(alaligne)`
`ent x=21`
`afficher (z)`
`ent y`
`x=x+1`
`y=6`
`afficher (x+19)`
`ent z`
 	
FIN
			`15`
			`41`

ALGO ent x= 2 ent y=1+x ent z=2 afficher(alaligne) afficher(2*x-y) afficher (x+z) x=23 FIN			25 43
ALGO ent x ent y=7 ent z=20 afficher(x+13) afficher (x) afficher (x+2) afficher(alaligne) afficher(alaligne) x=2 x=6 x=x+z x=x+y FIN			15 33 15

Exercice 3 Somme

a. Ecrivez, en pseudo-code un algorithme qui déclare deux entiers a et b, leur affecte une valeur, calcule leur somme qu'elle stocke dans une troisième variable c, puis affiche cette somme.
b. Modifier votre algorithme de manière à n'utiliser que deux variables (en supprimant c)
c. Modifier votre algorithme de manière à permettre à l'utilisateur de saisir la valeur des 2 entiers.

Exercice 4 Somme et moyenne

a. Ecrivez un algorithme qui affiche la somme et la moyenne de 4 notes saisies par l'utilisateur.
b. Modifiez ensuite votre programme pour n'utiliser que deux variables. Pourquoi est-il impossible de n'utiliser qu'une variable ?

Exercice 5 Echange du contenu de 2 variables

a. Ecrivez un algorithme qui
   - déclare et initialise deux variables x et y de type ENTIER et leur affecte une valeur (Par exemple 3 et 5)
   - Echange le contenu des deux variables
b. Idem sans utiliser de variable temporaire

Exercice 6 Troncature d'un reel positif

a. Ecrivez un algorithme (sans utiliser le type CHAINE) qui tronque un nombre réel – supposé positif - saisi par l'utilisateur et n'affiche que les 3 premiers chiffres après la virgule. Par exemple, si le nombre saisi est 2,3416, l'algorithme doit afficher 2,341
b. Modifier votre algorithme de manière à ce qu'il affiche le nombre arrondi 3 chiffres après la virgule : il affiche alors 2,342

Remarque : vous aurez besoin de la fonction partie entière

Exercice 7 Conversion heures/secondes

Ecrivez, en pseudo-code un algorithme qui convertit :

a. 3 nombre (heures, minutes, secondes) saisis par l'utilisateur en un nombre total de secondes, et qui affiche le résultat
b. un nombre de secondes (saisi par l'utilisateur) en (heures, minutes, secondes), et qui affiche le résultat. Par exemple, 7405 sec valent 2h 3 min 25 sec

B. Structures conditionnelles

Instructions conditionnelles

Pseudo code	Processing	C

SI expr_log
ALORS
instruction1
instruction 2
SINON
instruction 3
instruction 4
FSI
		if (expr_log){
		instruction1
		instruction2
		} else {
		instruction3
		instruction4
		}

où *expr_log* désigne une expression logique (à valeur booléenne) évaluable

Fonctionnement : si l'expression *expr_log* vaut

- VRAI, ce sont les instructions 1 et 2 qui sont exécutées
- FAUX ce sont les instructions 3 et 4 qui sont exécutées

Exercices

Exercice 7 : Simulez les algorithmes ci-dessous :

```
ALGO test
      ENT p, r, s
DEBUT
      p=4
      r=3*p-3
      p=r-3
      s=p+r/2
      SI s<p
            ALORS s=p
            SINON p=r
      FSI
      Afficher(p, s, r)
FIN test
```

```
ALGO test2
      ENT p, r, s
DEBUT
      p=-1
      r=3
      s=4
      SI 1<p OU p<3
            ALORS SI s!=p
                        ALORS s=p+s
                        SINON p=s*2
                  FSI
                  s=s-2
            SINON s=r+s
      FSI
      Afficher( p, s, r)
FIN test2
```

Exercice 7.5 : séquencement d'instructions

Dans le squelette de code ci-dessous, vous devez placer une ligne (et une seule) dans les 4 cases du code suivantes. Toutes les lignes doivent être utilisées. Le code ainsi obtenu doit **afficher *false* sur la console, et uniquement cela.**

x=x+y
x=x-y
afficher(x<y)
afficher (y==2)

ALGO

ENT x=1,y=2

SI(x>y) ALORS

SINON

FSI

Exercice 8 Jury

a. Ecrivez, en pseudo-code un algorithme qui demande à l'utilisateur d'entrer une note sur 20 et qui lui dit s'il a été reçu à l'examen (note supérieure ou égale à 10) ou bien s'il a échoué.
b. Modifiez votre programme pour qu'il discrimine plus finement les cas suivants :
   - note strictement inférieure à 10: échec
   - note comprise entre 10 et 12: cas de jury
   - note supérieure ou égale à 12: validé

Exercice 9 Maximum de 3 entiers

Ecrivez, en pseudo-code un algorithme qui affiche le plus grand des 3 nombres saisis par l'utilisateur. Combien votre algorithme contient-il de SI et d'opérateurs de comparaison ?

Exercice 10 Le jour suivant

On se propose, étant donnée une date (jour et mois seulement) de calculer la date du lendemain.

1. Ecrivez, en pseudo-code un algorithme qui demande à l'utilisateur le numéro d'un mois (entre 1 et 12), puis le numéro d'un jour et qui affiche ensuite le numéro du jour et celui du mois du jour suivant. On suppose que le numéro du jour et du mois sont corrects. Par exemple :
   > Quel mois ? 2
   > Quel jour ? 28
   > Le jour suivant est le 1/3.

   On supposera que l'utilisateur entre un numéro de mois et un numéro de jour correct.
2. Idem en demander à l'utilisateur l'année et en prenant en compte les années bissextiles qui sont les années :
   - divisibles par 4 mais non divisibles par 100, ainsi que celles qui sont
   - divisibles par 400

Votre algorithme contient il du code dupliqué ?

Vous pouvez envoyer votre réponse à la question 10.b à baudoin2020@gmail.com. Les 2 réponses les plus courtes seront créditées d'un bonus. Mettez votre algorithme directement dans le corps du mail et indiquez de combien de caractères se compose votre réponse.

Un algorithme bien écrit :

1. *Utilise des noms **éloquents** pour les variables **(tmp, moy, max, …)***
2. *Minimise le code dupliqué*
3. *Evite le code illisible (trop de SI imbriqués ou trop de combinaison logique de conditions, …)*

Chapitre 2

Boucles

A. Structures itératives

1. Boucle ITER

Itérations

Pseudo code	Processing	C

ITER
SORTIRSI expr_log 0
instruction1
SORTIRSI expr_log 1
instruction2
instruction3
SORTIRSI expr_log 2
instruction4
SORTIRSI expr_log 3
FITER
	while (expr_log0){
	instruction1
	instruction2
	instruction3
	instruction4
	}	

	do{ instruction1 instruction2 instruction3 instruction4 }while(expr_log3) ;	

Fonctionnement : Les instructions *instructionX* de la boucle sont répétées en boucle, dans l'ordre tant qu'aucune conditions de sortie *expr_log* n'est valide. Il faut au minimum un SORTIRSI dans une boucle ITER, le reste est facultatif.

Exemple :.

```
ALGO ex
ENT n
n=4
ITER
      n=n+3
      SORTIRSI    n > = 15
      Afficher (n)
FITER
FIN ex
```

Console : 71013

Exemple : filtre de saisie.

```
ALGO Filtre de saisie
ENT note
Afficher ("Entrez une note comprise entre 0 et 20 ")
ITER
      saisir(note)
      SORTIRSI note > = 0 ET note <=20
      Afficher ("Erreur, recommencez")
FITER
Afficher (" Votre note est bien comprise entre 0 et 20 : ", note)
FIN Filtre de saisie
```

2. Boucle POUR

Itérations

Pseudo code	Processing	C

POUR(instr_init; expr_log; instr)
  instruction1
  instruction2
  instruction3
  instruction4
FPOUR
	for(instr_init; expr_log;instr){
	  instruction1
	  instruction2
	  instruction3
	  instruction4
	}	

où :

- *instr_init* désigne l'instruction assertée avant toute itération
- *expr_log* désigne une expression logique evaluable qui doit être vérifiée pour rentrer dans la prochaine itération
- *instr* désigne l'instruction assertée à la fin de chaque itération

Comportement : Les instructions actions 1, action 2, … sont exécutées en boucle tant que *expr_log* vaut VRAI. Avant de recommencer une nouvelle itération, l'instruction *instr* est exécutée

Exemple 0 :

```
int i=0;
for (println("Tchiky"); i<2; println("tchak") ) {
  println("boum");
}
```

Exemple : Répéter 3 fois l'affichage de la chaine '' bonjour''

```
ALGO AffRep
Ent i
Afficher ("avant ")
POUR (i=0 ; i<3 ; i = i+1)
      Afficher ("bonjour")
FPOUR
Afficher ("après ")
FIN AffRep
```

Exemple : Afficher les nombres de 1 à 23

```
ALGO Aff
Ent i
POUR (i=1 ; i<=23 ; i = i+1)
      Afficher (i)
```

```
FPOUR
FIN Aff
```

3. EQUIVALENCE boucle POUR / boucle ITER

```
POUR (instr_init ; expr_log ; instr)

      actions1
      actions2
      actions3

FPOUR
```

⟺

```
instr_init

ITER
      SORTIRSI NON(expr_log )

      actions1
      actions2
      actions3

      instr
FITER
```

B. Exercices en pseudocode

Exercice 11 Simulation d'algorithme
Simulez le comportement des algorithmes suivants :

```
ALGO iter
      ENT i
DEBUT
      POUR (i=0; i<10; i=i+1)
            Afficher(i+2)
            Afficher (a_la_ligne)
      FPOUR
      Afficher("bonjour")
FIN iter
```

```
ALGO iter2
      ENT i
DEBUT
      POUR (i=1; i<10; i=i+i)
            Afficher(i)
            Afficher (a_la_ligne)
      FPOUR
FIN iter2
```

Exercice 11.5 Simulation d'algorithme
Simulez le comportement des algorithmes suivants :

```
ALGO iter
      ENT i
DEBUT
      POUR (i=0; i>10; i=i+1)
            Afficher(i)
            Afficher (a_la_ligne)
      FPOUR
      Afficher("bonjour")
FIN iter
```

```
ALGO iter2
      ENT i
DEBUT
      POUR (i=0; i<10; i=i+i)
            Afficher(i)
            Afficher (a_la_ligne)
      FPOUR
FIN iter2
```

Exercice 12 Simulation d'algorithme
Simulez le comportement des algorithmes suivants :

```
ALGO iter3
      ENT i,j
DEBUT
      POUR (i=0; i<=3; i=i+1)
            j=i*i+1
            Afficher(j)
            Afficher (a_la_ligne)
      FPOUR
      Afficher(i)
FIN iter3
```

```
ALGO iter4
      ENT i,j=0
DEBUT
      POUR (i=1; j<5; i=i+1)
            i=i-1
            SI i!=0
                  ALORS j=j+1
                  SINON j =5
            FSI
            Afficher(j)
            Afficher (a_la_ligne)
```

	FPOUR Afficher(i) FIN iter4

Exercice 14 Règle

Ecrivez, en pseudo-code un algorithme qui demande une longueur et affiche une règle de cette longueur avec des tirets :

>Longueur ? 53

Modifier votre programme pour qu'il affiche des graduations suivant un intervalle également demandé à l'utilisateur :

>Longueur ? 53
>Intervalle ? 10
|---------|---------|---------|---------|---------|---

Exercice 15 Moyenne d'une suite

Ecrivez, en pseudo-code un algorithme qui permet la saisie des éléments d'une suite de nombres entiers positifs. Pour indiquer que la saisie des termes est terminée, l'utilisateur entrera un nombre négatif. L'algorithme doit alors afficher la somme de ces éléments, leur nombre, et leur moyenne.

Par exemple, si les saisies utilisateur sont : 2, 5, 4, -1, votre code doit afficher :

Il y a 3 valeurs, leur somme vaut 11 et la moyenne entière vaut 3

Exercice 16 Maximum d'une suite

Idem en se limitant à l'affichage de l'élément le plus grand de la suite

Exercice 17 Minimum d'une suite

Idem en se limitant à l'affichage de l'élément le plus petit de la suite

Exercice 18 Entiers répétés

a. Ecrivez un algorithme qui met en œuvre l'interaction suivante :
   - l'utilisateur saisit des entiers
   - l'algorithme affiche le message « répétition » à l'écran si 2 entiers consécutivement saisis sont égaux
   - la saisie prend fin au premier nombre négatif

b. Modifiez votre algorithme de manière à ce qu'il affiche une seule fois le message quand un entier est répété plus de 2 fois de suite

Exercice 19 Itération d'un opérateur

a. Multiplication

Ecrivez, en pseudo-code un algorithme qui, étant donnés deux nombres entiers *a* et *n*, calcule *a* * *n* sans utiliser l'opérateur *

b. Puissances

Ecrivez, en pseudo-code un algorithme qui, étant donnés deux nombres entiers *a* et *n*, calcule *a* à la puissance *n* sans utiliser l'opérateur ^

c. Factorielle

Ecrivez, un algorithme qui, étant donnés un nombre entier positif *n* calcule *n ! sans* utiliser l'opérateur !. Pour rappel, 5!=5*4*3*2*1=120

Exercice 20 Somme des premiers entiers positifs

Ecrivez, en pseudo-code un algorithme qui calcule et affiche la somme des n premiers nombres entiers, où n est un entier choisi par l'utilisateur. Par exemple, pour n = 5, 5+4+3+2+1=15

Exercice 21 Nombres premiers

Ecrivez, en pseudo-code un algorithme qui affiche si un nombre saisi par l'utilisateur est un nombre premier. On rappelle qu'un nombre premier est un nombre entier qui a exactement 2 diviseurs : 1 et lui-même. On rappelle également qu'un entier a est divisible par un autre entier b si le reste de la division euclidienne de a par b vaut 0, c'est-à-dire si a%b=0.

Exercice 22 Fibonacci

Ecrire un algorithme qui étant donné un entier n saisie au clavier, permet de calculer le n-ième terme de la suite de Fibonacci définie par :

$$U_n = U_{n-1} + U_{n-2}\ \forall n > 1 \text{ avec } U_0 = 0 \text{ et } U_1 = 1$$

C. Doubles boucles

Exercice 23 Simulation d'algorithme

Simulez le comportement de l'algorithme suivant :

```
ALGO doubleBoucle
      ENT i,j
DEBUT
      POUR (i=0; i<3; i=i+1)
            POUR (j=0; j<3; j=j+1)
                  Afficher (i,j)
            FPOUR
      FPOUR
FIN doubleBoucle
```

Correction : 000102101112202122

Modifiez le code de manière à produire la console suivante :

000102

101112
202122

Exercice 23.5 Simulation d'algorithme
Simulez le comportement de l'algorithme suivant :

```
int i, j;
for (i = 0; i<4; i=i+1) {
  for (j = 0; j<4; j=j+1) {
    if ((i+j)==2)println(i, j);
    if ((i+j)==4)println(j, i);
  }
}
```

Exercice 24 Table de multiplication

a. Ecrivez, en pseudo-code un algorithme qui affiche une table de multiplications (double boucle obligatoire) sous la forme:

```
0 0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10
0 2 4 6 8 10 12 14 16 18 20
0 3 6 9 12 15 18 21 24 27 30
0 4 8 12 16 20 24 28 32 36 40
0 5 10 15 20 25 30 35 40 45 50
0 6 12 18 24 30 36 42 48 54 60
0 7 14 21 28 35 42 49 56 63 70
0 8 16 24 32 40 48 56 64 72 80
0 9 18 27 36 45 54 63 72 81 90
0 10 20 30 40 50 60 70 80 90 100
```

b. Améliorez votre algorithme en séparant les termes de la table par le caractère *barre verticale* « | », et en alignant les colonnes

```
|  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
|  0 |  1 |  2 |  3 |  4 |  5 |  6 |  7 |  8 |  9 | 10 |
|  0 |  2 |  4 |  6 |  8 | 10 | 12 | 14 | 16 | 18 | 20 |
|  0 |  3 |  6 |  9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
|  0 |  4 |  8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
|  0 |  5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|  0 |  6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
|  0 |  7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
|  0 |  8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
|  0 |  9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
|  0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |100 |
```

Exercice 25 Etoiles

a. Ecrivez, en pseudo-code un algorithme qui demande à l'utilisateur un entier n, puis affiche avec des étoiles un rectangle de côté n. contrainte : on utilisera exclusivement des boucles POUR, avec incrémentation positive du compteur de boucle à partir de 0.

>Cote ? 4

```
****
****
```

```
****
****
```

Correction

```
ALGO etoiles
      ENT i,j, n
afficher ("Cote ? ")
saisir(n)
POUR (i=0; i<n; i=i+1)
      POUR (j=0; j< n  ; j=j+1)
            Afficher ("*")
      FPOUR
      afficher (alaligne)
FPOUR
```

b. Modifier votre algorithme pour afficher un triangle:

```
*
**
***
****
```

Contrainte : vous ne pouvez modifier que la partie grisée de la correction

c. Même question avec la pointe dans l'autre sens:

```
****
***
**
*
```

d. Même question :

```
****
 ***
  **
   *
```

Exercice 26 Rectangle

Même exercice avec le rectangle :

>Largeur ? 9

>Hauteur ? 3

```
*********
|       |
|       |
|       |
*********
```

Exercice 27 Entiers parfaits

Ecrivez, en pseudo-code un algorithme qui détermine les entiers parfaits inférieurs à 100. Un entier naturel est parfait s'il est la somme de ses diviseurs *propres*, c'est-à-dire différents de lui-même. Ainsi 6 est un nombre parfait car 6 = 1 + 2 + 3.

Exercice 28 Suite aliquote

Ecrivez, en pseudo-code un algorithme qui détermine la suite aliquote d'un entier saisi par l'utilisateur. Une suite aliquote est une suite d'entiers dans laquelle chaque nombre est la somme des diviseurs propres de son prédécesseur. Quand la suite atteint 1, elle s'arrête car 1 ne possède pas de diviseur propre. Par exemple, pour une suite commençant à U_0=10. Les diviseurs propres de 10 sont 1, 2 et 5 donc U_1=1+2+5=8. Les diviseurs propres de 8 sont 1, 2 et 4 donc U_2=1+2+4=7. Or 7 ne possède qu'un diviseur propre 1 donc U_3=1

Dans un algorithme bien écrit :

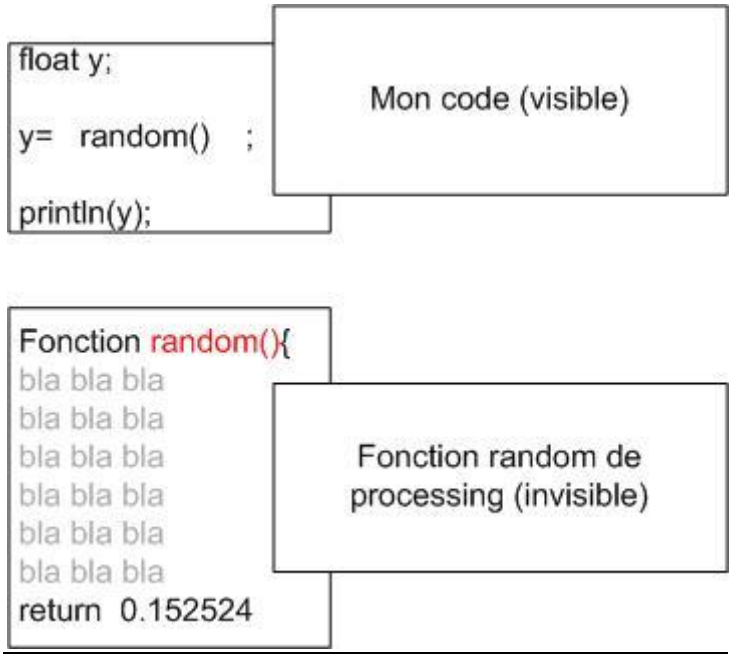
1. *La valeur du compteur n'est jamais modifié dans le corps d'une boucle POUR*
2. *Les compteurs commencent à 0 et leurs valeurs sont croissantes (sauf raison particulière)*

Chapitre 3
Fonctions

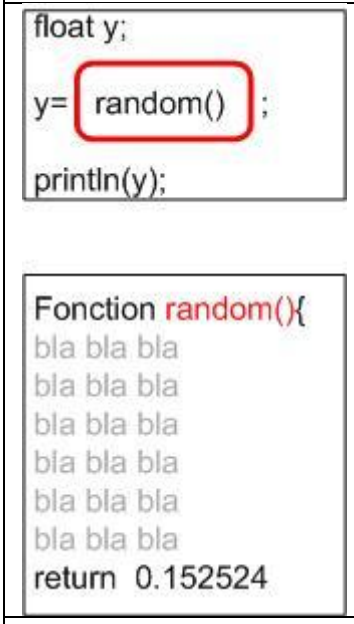
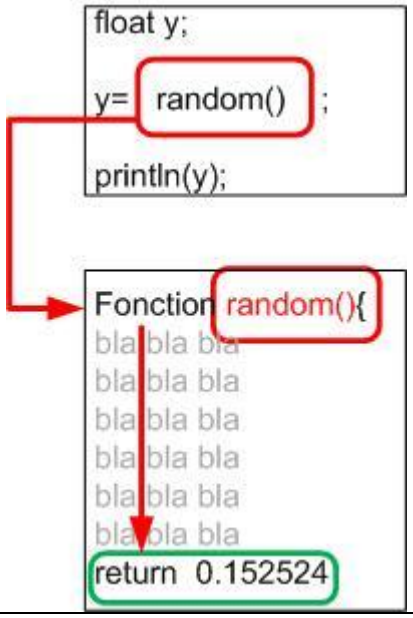
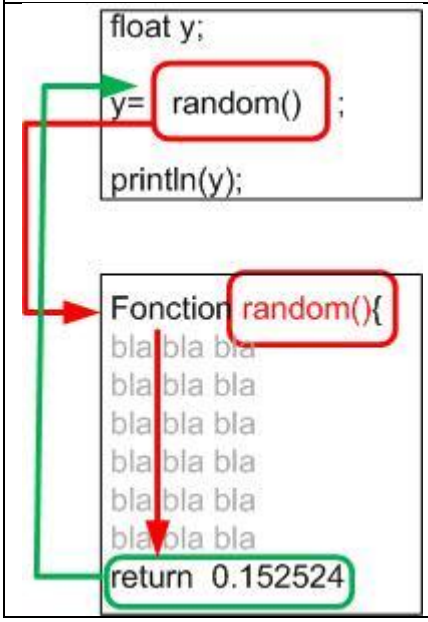
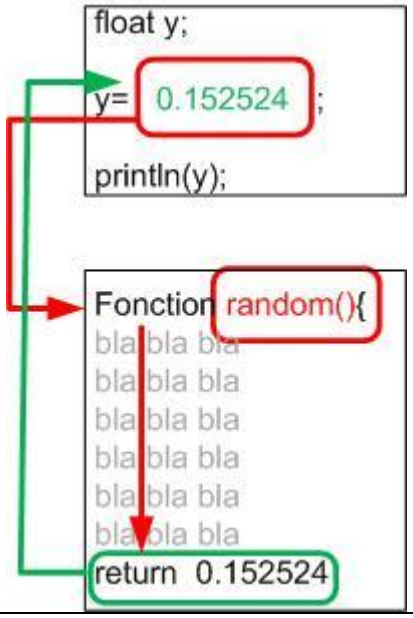
A. Principe

Les fonctions permettent d'organiser votre code. On va voir comment le code d'un programme peut être écrit à 2 endroits différents

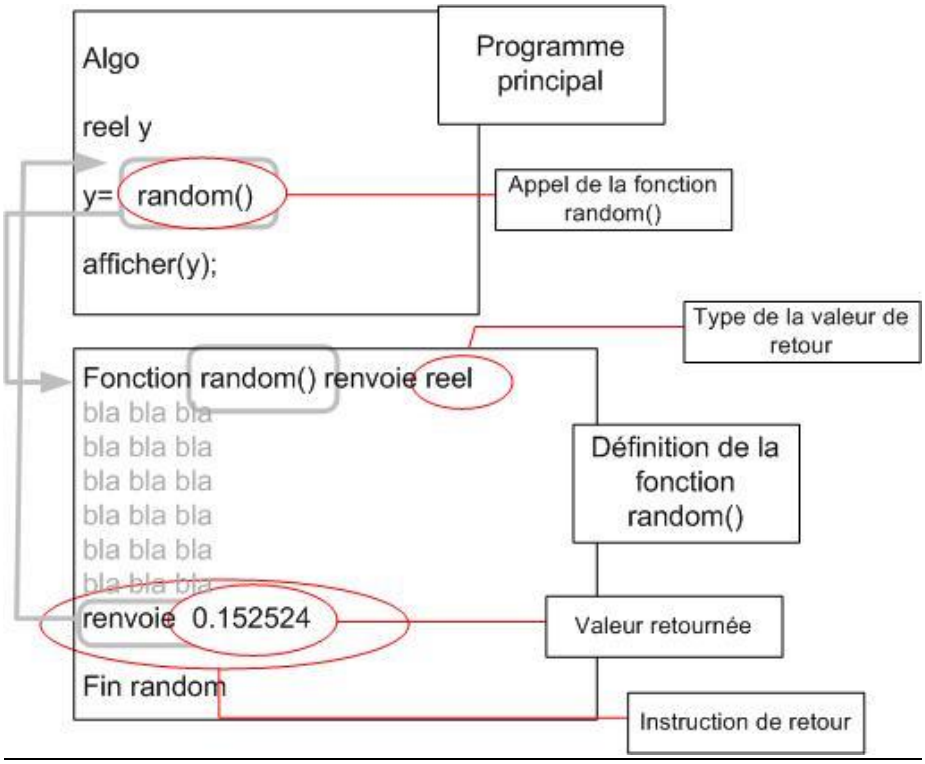
Exemple de la fonction random() de Java



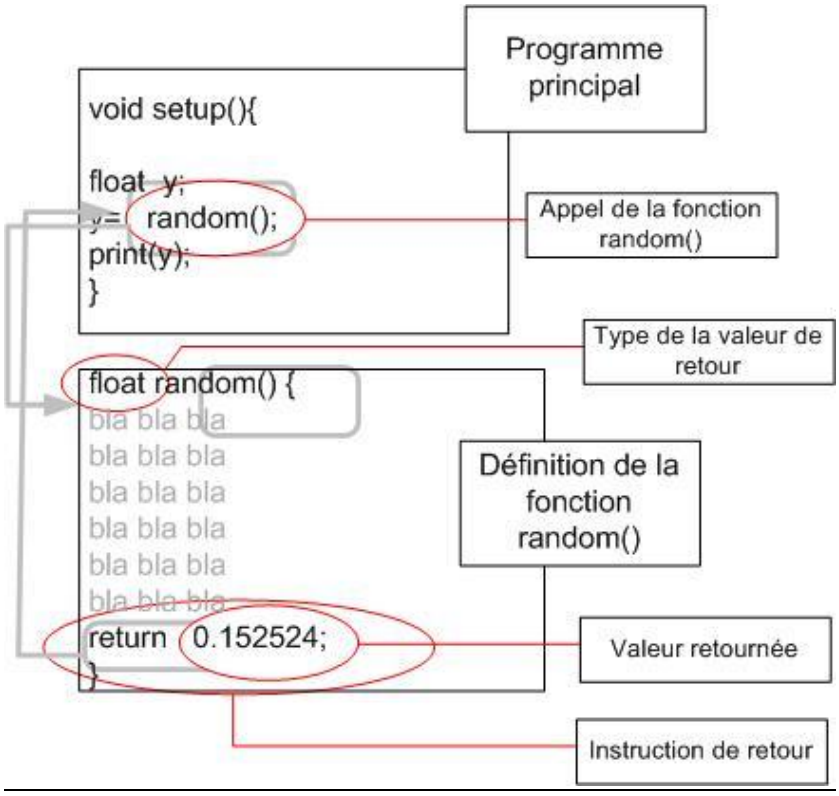
Coordination

1. Appel de la fonction random	2. Exécution du corps de la fonction random
	
3. Retour de la valeur	4. Remplacement de l'appel de la fonction par la valeur retournée
	

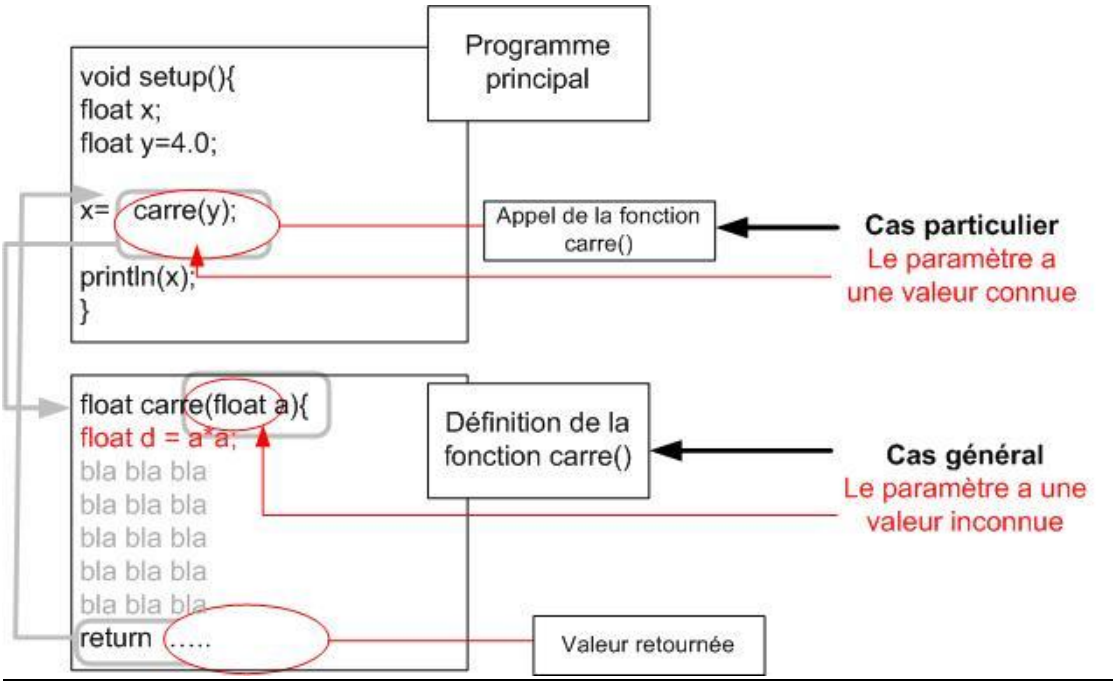
Terminologie et syntaxe Pseudo Code



Terminologie et syntaxe Processing



Cas des fonctions avec paramètres : la fonction carré(float x)



Portée des variables (scope) :

Le nommage des variables n'est valuable (visible) que dans la fonction où les variables ont été **declares**

Définition (declaration) des fonctions

Pseudo code	Processing	C

Fonction moy(Ent a, Ent b) renvoie Ent
renvoie (a+b)/2 ;
Fin moy
	int moy(int a, int b){
	return (a+b)/2 ;
	}	

Appel des fonction

Pseudo code	Processing	C
	`moy(2, 5)`	

Donc

Quand le code est réparti à 2 endroits différents, il faut que

- L'exécution des 2 portions de code soient coordonnées
- Ces portions de codes puissent se communiquer des valeurs

Exercice : Dire si les programmes suivants compilent. Sinon, les corriger.

`void setup() {`
`println(f(2.6));`
`}`
 	
`float f(int a) {`
`  //blablabla`
`  return 0.1;`
`}`
	`void setup() {`
	`println(f(2, 3.0));`
	`}`
	
	`float f(float a, int b) {`
	`  //blablabla`
	`  return 0.1;`
	`}`

`void setup() {`
`  float x=5.0;`
`  int y = f(x);`
`}`
 	
`float f(float a) {`
`  float b=0.1;`
`  //blablabla`
`  return b;`
`}`
	`void setup() {`
	`  int x=2;`
	`  int y = f(f(x));`
	`}`
	
	`int f(int a) {`
	`  int b;`
	`  //blablabla`
	`  return b;`
	`}`

`void setup() {`
`  int x=2;`
`  int c=2;`
`  int y = f(x);`
`}`
 	
`int f(int a) {`
`  int b=c;`
`  //blablabla`
`  return b;`
`}`
	`void setup() {`
	`  int x=2;`
	`  float y = f(x);`
	`}`
	
	`void f(int a) {`
	`  //blablabla`
	
	`}`

`void setup() {`
`  int x=2;`
`  f(x);`
`}`
 	
`void f(int a) {`
`  float b;`
`  //blablabla`
`  return b;`
`}`
	`void setup() {`
	`  int x=2;`
	`  f(x);`
	`}`
	
	`int f(int a) {`
	`  int b = 2;`
	`  //blablabla`
	`  return b;`
	`}`

Exercice : Ecrire la définition d'une fonction toto qui est appelée dans le code ci-dessous (sans conversion de type)

Pseudo code	Processing

Algo
  ent x=2
  reel c=2.0
  car y = toto(x, c)
Fin
 	
Fonction toto( )renvoie
  //blablabla
Fin toto
	void setup() {
	  int x=2;
	  float c=2.0;
	  char y = toto(x, c);
	}
	
	  toto( ){
	  // blabla
	}

A. Exercices en pseudocode

Exercice 1 Minimum de 2 ou 3 entiers

1. Ecrivez, en pseudo-code une fonction *min2* qui prend en paramètres 2 entiers a et b et qui renvoie le plus petit des 2. Ecrire l'algorithme de test correspondant.
2. Ecrivez une fonction *min3* qui prend en paramètres 3 entiers a , b et c et qui renvoie le plus petit des 3. contrainte : *min3* ne fera aucune comparaison mais utilisera la fonction min2

Exercice 2 Fonction syr

a. Ecrivez, en pseudo-code une fonction *syr* qui prend en paramètre un entier positif n, et qui renvoie un entier dont la valeur vaut :
   a. n/2 si n est pair
   b. 3*n+1 sinon

b. On peut utiliser la fonction syr de manière itérée (à partir d'une valeur initiale) . Par exemple, pour la valeur initiale 15, l'application itérée de la fonction donne :

   15 →46→23→70→35→106→53→160→80→40→20→10→5→16→8→4→2→1

   La suite de ces nombres forme ce qu'on appelle la suite *syracuse 15*
   - A la main, calculer les suites *syracuse 11 et syracuse 27*
   - Ecrivez, en pseudo-code une fonction *suiteSyr* qui prend en paramètre un entier p (la valeur initiale) et affiche la suite Syracuse<p> correspondante.

c. On appelle

   - *temps de vol* le nombre d'itérations avant de tomber sur 1. Il est de 17 pour la suite de Syracuse 15 et de 46 pour la suite de Syracuse 127

   - *temps de vol en altitude* : c'est nombre d'itérations nécessaires pour repasser en dessous de la valeur de départ. Il est de 10 pour la suite de Syracuse 15 et de 23 pour la suite de Syracuse 127

- l'altitude maximale : c'est la valeur maximale de la suite.Elle est de 160 pour la suite de Syracuse 15 et de 4 372 pour la suite de Syracuse 127.

➔ Ecrire les fonctions tempsDeVol, tempsDeVolEnAlti, altitudeMax en choisissant un paramètrage adapté

Exercice 3 Plus grand commun diviseur

Ecrivez, en pseudo-code une fonction *pgcd* qui prend en paramètres 2 entiers a et b et qui renvoie leur plus grand commun diviseur. Par exemple, *pgcd*(18,24) doit renvoyer 6.

Exercice 4 Modulo

Ecrivez une fonction qui prend en paramètre 2 entiers a et b et qui renvoie un entier valant a%b (sans utiliser l'opérateur modulo, ni la division entière).

Exercice 5 Primalité

a. Ecrivez une fonction Prem qui prend en paramètre un entier positif a et qui renvoie un booléen valant VRAI si a est un nombre premier, et FAUX sinon.
b. Conjecture de Goldbach : La conjecture affirme que tout nombre entier pair strictement supérieur à 3 peut s'écrire comme la somme de deux nombres premiers. Ecrire une fonction GOLDBACH qui prend en paramètre un entier pair, et qui affiche les 2 nombres premiers dont cet entier est la somme.

Exercice 6

1. Ecrire une fonction *nbchiffres* qui prend en paramètre un entier (supposé positif) et qui renvoie le nombre de chiffes de ce nombre.
2. Ecrire une fonction *divisionEntiere* qui prend en paramètre 2 entiers a et b, et qui renvoie la division entière de a par b. Interdit d'utiliser l'opérateur « / ». Par exemple, si a vaut 30 et b vaut 12, la fonction doit renvoyer 2. On pourra se limiter aux cas où a et b sont 2 entiers strictement positifs

Exercice 7

1. Ecrire une fonction somDiviseurs qui prend en paramètre un entier n et qui renvoie un entire qui vaut la somme de ses diviseurs propres (c'est à dire les diviseurs different de lui meme). Par exemple, somDiviseurs(10) doit renvoyer 8 car 1+2+5=8
2. Ecrire une fonction entierParfait qui prend en paramètre un entier n et qui renvoie un booléen qui vaut :
   a. true si n est un entier parfait. Un entier naturel est parfait s'il est la somme de ses diviseurs propres, c'est-à-dire différents de lui-même. Ainsi 6 est un nombre parfait car 6 = 1 + 2 + 3. Il existe 3 entiers parfaits inférieurs à 1000 : 6, 28 et 496.
   b. false sinon.
3. Ecrire une fonction entiersAmis qui prend en paramètre 2 entiers a et b et qui renvoie un booléen qui vaut :
   a. True si les entiers a et b sont amis. Deux nombres sont amis si la somme des diviseurs propres de l'un est égale à l'autre et réciproquement.
   b. False sinon

Par exemple, les nombres amis inférieurs à 10k

220	284
1184	1210
2620	2924
5020	5564
6368	6632

Exercice 8 Conjecture de Polya

La conjecture affirme que pour tout entier N, dans la décomposition en facteurs premiers des entiers naturels inférieurs à N, il y a d'avantage (ou autant) de décompositions avec un nombre impair de facteurs que de décompositions avec un nombre pair de facteurs.
Par exemple, avec N=12

Entier	Decomposition en facteurs premiers	Taille de la decomposition	Parité de la taille de la decomposition
1		0	paire
2	2	1	impaire
3	3	1	impaire
4	2*2	2	paire
5	5	1	impaire
6	2*3	2	paire
7	7	1	impaire
8	2*2*2	3	impaire
9	3*3	2	paire
10	2*5	2	paire
11	11	1	impaire
12	2*2*3	3	impaire

Pour les entiers inférieurs à N=12, on a donc 5 pairs et 7 impairs, on a bien **"nb impairs " >= "nb pairs"**, la conjecture est vérifiée pour n = 12

Exercice 9 Entiers palindromes

Ecrire une fonction isPal qui prend en paramètre un entier positif n et qui renvoie un booléen qui vaut :

- true si l'entier est palindrome, par exemple, 1221 ou 14541
- false sinon

Exercice 10 Somme de palindromes

Ecrire une fonction isSomPal qui prend en paramètre un entier positif n et qui affiche s'il vaut la somme de 2 entiers palindromes. par exemple, pour n = 152, on a 152 = 11 + 141. Que remarque t on ? Même questions avec la somme de 3 palindromes.

Chapitre 3
Tableaux

A. Tableaux

Exemple d'un tableau tab de 8 entiers.

On a tab[0]=4, tab[7]=-1, tab[tab[6]]=8

Par contre, tab[-1], tab[9], tab[8] , hors plage, déclenchent une erreur

indices	0	1	2	3	4	5	6	7
valeurs	4	6	8	-4	0	0	2	-1

Déclaration / déclaration+initialisation

Pseudo code	Processing	C
TABLEAU tab ENT[10]	int[] tab = new int[10]	int tab[10]
TABLEAU tab = {15,8,11,0, 0}	int[] tab = {15,8,11,0,0}	int tab[10]={15, 8, 11}

Accès

Pseudo code	Processing	C
tab[2]		

Remarque : les indices d'un tableau de longueur N vont de 0 à N-1.

Boucle de parcours des cases d'un tableau	

`ENT[] tab = {2, 4, 6, 8}`
`Ent i`
`Pour (i = 0 ; i< 4 ; i++)`
`  Afficher(tab[i])`
`FPOUR`
	`ENT[] tab = {2, 4, 6, 8}`
	`Ent i`
	`Pour (i = 0 ; i< taille(tab) ; i++)`
	`  Afficher(tab[i])`
	`FPOUR`

Fonction prenant un tableau en paramètre	

	`Fonction AffTab(Ent[] tab) renvoie rien`
	`Pour (i = 0 ; i< taille(tab) ; i++)`
	`  Afficher(tab[i])`
	`FPOUR`

B. Exercices en pseudo code

Exercice 1 Initialisation de tableaux

Ecrivez, en pseudo-code un algorithme qui initialise 3 tableaux de 10 entiers (en séparant la déclaration du tableau de son initialisation, en utilisant obligatoirement des boucles) :

- le premier tableau ne comporte que des 0
- le 2ème tableau comprend les 10 premiers entiers positifs dans l'ordre croissants : de 1 à 10
- le 3ème tableau commence à 20 et décroit de 2 en 2 jusqu'à 2.

Exercice 3 Simulation d'algorithme

Simulez le comportement de l'algorithme suivant :

```
ALGO tab1
        ENT i
        TABLEAU tab    = {15,8,11,12,10}
DEBUT
        POUR (i=0; i<taille(tab); i=i+1)
                tab[i]=tab[i]/2
        FPOUR

FIN tab1
```

Exercice 4 Simulation d'algorithme

Donner les valeurs du tableau tab à la fin des algorithmes suivants :

```
ALGO tab1
        ENT i
        TABLEAU tab   ENT[10]
        POUR (i=0; i<taille(tab); i=i+1)
                tab[i]=i*i
        FPOUR
FIN tab1
```

```
ALGO tab1.2
        ENT i
        TABLEAU tab   = {15,8,11,12,10}
        Ent som=tab[0]
        POUR (i=1; i<taille(tab); i=i+1)
                som=som+tab[i-1]
                tab[i]=som
        FPOUR
FIN tab1.2
```

```
ALGO tab2
        ENT i
        TABLEAU tab   = {15,8,11,12,10}
DEBUT
        POUR (i=0; i<taille(tab); i=i+1)
                tab[i]=tab[i/2]
        FPOUR
FIN tab2
```

```
ALGO tab3
        ENT i
        TABLEAU tab   = {15,8,11,12,10}
DEBUT
        POUR (i=taille(tab)-1;i>=0; i=i-1)
                tab[i]=tab[i/2]
        FPOUR
FIN tab3
```

Exercice 5 Moyenne des éléments d'un tableau

Écrire une fonction *moyTab* qui prend en entrée un tableau tab d'entiers et renvoie la moyenne (entière) des éléments de tab.

Exercice 6 Recherche simple dans un tableau

1. Ecrire une fonction *rechSimplePrem* qui prend en paramètre un tableau tab d'entiers et un entier n, et qui renvoie l'indice du tableau dont la valeur vaut n. S'il existe plusieurs indices, la fonction doit renvoyer le premier. Si la valeur ne se trouve pas dans le tableau, la fonction doit renvoyer -1.
2. Idem avec une fonction *rechSimpleDer* qui renvoie le dernier indice.

Exercice 7 Permutations

1. Ecrire une fonction *permut* qui prend en entrée un tableau d'entiers tab et 2 entiers i et j et qui permute les valeurs des cases d'indices i et j du tableau tab.
2. Ecrire un algorithme qui utilise la fonction permut sur un exemple simple de tableau.

Exercice 8 Test d'uniformité

Ecrire une fonction *uniformTab* qui prend en entrée un tableau d'entiers et qui renvoie VRAI si toutes les valeurs du tableau sont identiques et FAUX sinon

Exercice 9 Histogramme

Ecrivez, en pseudo-code un algorithme permettant de tracer l'histogramme d'une série de notes comprise entre 0 et 9. La saisie prendra fin quand l'entier -1 sera lu. L'histogramme sera affiché

verticalement en utilisant le caractère « * ». Par exemple, si la saisie comporte : 4 fois la note 9, 0 fois la note 8 et 8 fois la note 7 (et ainsi de suite) :

9 : ****
8 :
7 : ********

Exercice 10 Crible d'Eratosthène

Il est possible d'obtenir les nombres premiers compris entre 1 et n en utilisant la méthode du crible d'Eratosthène qui suit les étapes proposées ci-dessous.

a. On déclare un tableau nommé crible de n+1 entiers.
b. On initialise ce tableau en plaçant dans chaque case d'indice i la valeur i.
c. On met à 0 tous les éléments du tableau qui ne sont pas premier. Ces éléments sont identifiés comme des multiples de nombres premiers et calculés par propagation multiplicative (par définition, 1 n'est pas premier)

Par exemple, crible[2] vaut 2, donc on met à 0 crible[4], crible[6], crible[8], etc jusqu'à N car 4, 6 et 8 sont des multiples de 2. On passe ensuite à crible[3] qui vaut 3, donc on fait la même chose. crible[4] vaut 0 donc on ne fait rien, et on passe à crible[5], … A la fin des propagations multiplicatives successives, les nombres premiers correspondent aux indices des cases non nulles du tableau.

En utilisant cette méthode, écrire un algorithme qui affiche tous les nombres premiers inférieur à un entier N, fourni par l'utilisateur.

Exercice 11 Simulation d'algorithme

Déterminer la sortie du programme suivant (La fonction uniformeTab est celle de l'exercice 6)

```
ALGO exemple
        ENT i, x=0
        TABLEAU tab   = {15,8,11,12,10}

DEBUT
ITER
        SORTIRSI (uniformTab(tab)==VRAI)
                POUR (i=0 ; i< taille(tab); i++)
                        tab[i]=f(tab[i])
                FPOUR
                x=x+1

FITER
Afficher(x)
FIN exemple
```

```
FONCTION f(E ENT n) RENVOIE ENT
DEBUT
        SI n%2=0
                ALORS RENVOIE n/2
                SINON RENVOIE (n-1)/2
        FSI
FIN f
```

Exercice 12 Test de croissance

Ecrire une fonction *testCroiss* qui prend en entrée un tableau d'entiers et affiche s'il s'agit d'un tableau constant, croissant (non strictement), décroissant (non strictement) ou non trié.

Bonus : idem mais avec une fonction numérique (sur des décimaux énumérables) : La fonction numérique est codée en dur (elle n'apparait pas en tant que paramètre) dans la fonction à écrire. Bien entendu, on se limitera aux affichages suivants : les intervalles où la fonction est croissante, ceux où elle est décroissante, ceux où elle est constante.

Exercice 13a Recherche de doublons

Ecrire une fonction *rechDoublons* qui prend en entrée un tableau d'entiers tab et qui renvoie un booléen dont la valeur vaut VRAI si le tableau contient des doublons et FAUX sinon. Pouvez vous donner la complexité de l'algorithme (le nombre de comparaisons en fonction de n) ?

Exercice 13b Recherche d'un doublon exactement

Ecrire une fonction *rechDoublons* qui prend en entrée un tableau d'entiers tab et qui renvoie un booléen qui vaut :

- VRAI si le tableau contient exactement un doublon : C'est-à-dire si une seule aleur se répète et qu'elle ne se repete qu'une fois. Par exemple tab = {-3, 13, 9, 13, 17, 21},
- FAUX sinon. Par exemple
    - tab = {-3, 1, 5, 9, 13, 17, 21} : aucune valeur ne se répète
    - tab = {1, 5, 9, 1, 1, 21} : une valeur se répète plus de 2 fois
    - tab = {1, 5, 9, 1, 5, 21} : plusieurs valeurs se répètent

Exercice 13e Recherche des plus proches doublons

Ecrire une fonction *rechDoublonsProches* qui prend en entrée un tableau d'entiers tab et qui renvoie un entier qui vaut :

- la distance qui sépare les 2 doublons les plus proches. Par exemple si tab = {3, 4, 5, 1, 9, **3**, 6, 1, 4, **3**}, la fonction doit renvoyer 4 car il y a 4 cases entre le doublon de 3 en gras.
- 0 sinon

Exercice 13d Recherche du plus petit doublon

Ecrire une fonction *rechMinDoub* qui prend en entrée un tableau d'entiers **positifs** tab et qui renvoie un entier qui vaut :

- le plus petit entier qui se répète au moins 1 fois Par exemple l'entier 2 si tab = { **2**, 3, 4, **13**, 9, **2**, **13**, 17, 21},
- -1 si il n'y a pas de doublon.

Exercice 13c Recherche du premier doublon

Ecrire une fonction *rechPremDoublons* qui prend en entrée un tableau d'entiers tab (supposés positifs) et qui renvoie un entier qui vaut

- la valeur qui double en premier dans une lecture des indices croissants du tableau. Par exemple, si tab={3, 6, **2**, 7, **2**, 3, 3, 6, 1}, la fonction doit renvoyer 2 (et ni 6, ni 3)
- -1 sinon.

Exercice 14 Recherche dichotomique

Ecrivez une fonction *rechDich* qui permet de rechercher un entier n dans un tableau T d'entiers trié par ordre croissant. La fonction doit renvoyer l'indice du tableau qui a pour valeur n ou -1 si la valeur n'est pas dans le tableau. Vous procéderez selon le principe suivant :

- On tient en permanence deux indices gauche et droite tels que T[gauche] <= n <= T[droite].

- À chaque étape, on prend le milieu de l'intervalle [gauche, droite] et on le compare à n.
- En fonction du résultat, soit on affiche l'indice du milieu, soit on met à jour gauche ou droite et on continue à chercher n dans le sous-tableau correspondant.

Que se passe t il quand la valeur ne se trouve pas dans le tableau ?

Exercice 15 Polynômes

Les polynômes d'une variable à coefficients réels ou décimaux s'écrivent :

$$P(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_n * x^n = \sum_{i=0}^{n} a_i * x^i$$

On peut représenter les polynômes à l'aide de tableaux de nombres réels où sont stockés les coefficients. Il faudra un tableau de n + 1 élément pour représenter un polynôme de degré n. Par exemple, on peut représenter le polynôme $P(x) = 3 + 7x^2 + 4x^3$ avec un tableau de réels {3,0,7,4}

a. Ecrire une fonction *eval* qui prend en paramètre un tableau tab de réels qui représente un polynôme P et un réel x, et qui renvoie la valeur de P en x
b. Ecrire une fonction qui additionne deux polynômes ainsi représentés.
c. Ecrire une fonction qui calcule le polynôme dérivé
d. Ecrire une fonction qui multiplie deux polynômes.
e. Ecrire une fonction qui compose deux polynômes.

Exercice 15.5 Elements les plus proches dans un tableau

Ecrire une fonction qui prend en paramètre un tableau d'entier et qui la distance entre les 2 éléments les plus proches du tableau. Par exemple,

T = { 4, 7, 9, 1, 7 } la fonction doit renvoyer 0 car 7-7=0
T = { 4, 7, 9, 1, 17 } la fonction doit renvoyer 2 car 9-7=2
On pourra utiliser la fonction valeur absolue **abs(int a)** : abs(9-7)=abs(7-9)=2

Exercice 16 Tableau et comparaison case/moyenne

Ecrire une fonction qui prend en paramètre un tableau d'entier et qui renvoie VRAI si l'une des cases du tableau vaut la moyenne de 2 autres cases (d'indice différent).

Par exemple, si T = { 4, 7, 9, 1, 10, 7 } la fonction doit renvoyer vrai car (1+7)/2=4

Exercice 17 Tableau et comparaison case/différence

1. Ecrire une fonction qui prend en paramètre un tableau d'entier et qui renvoie VRAI si l'une des cases du tableau vaut la différence de 2 autres cases. Par exemple, si T = { 4, 7, 9, 1, 5 } la fonction doit renvoyer vrai car 9 – 5 = 4
2. Modifier votre fonction de manière à ce qu'elle affiche (ou renvoie) la chaine *'' oui car 9 – 5 = 4'' ou ''non''*

Exercice 18 Sous tableau de somme maximale

Le but de l'exercice est de déterminer pour un tableau d'entiers quelconques, le sous tableau dont la somme des éléments est la plus grande.

0	1	2	3	4	5	6
-6	12	-7	0	14	-7	5

Par exemple, si T = { -6,12,-7,0,14,-7,5 } alors le sous tableau à déterminer est T'= { 12,-7,0,14 } car 19 =est la somme partielle de cases adjacentes la plus grande (par exemple, la somme du sous tableau T'' = { -6,12,-7 } vaut 1, ce qui est moins que 19.

- Que vaut le sous tableau si le tableau ne contient que des entiers positifs ?
- Que vaut le sous tableau si le tableau ne contient que des entiers négatifs ?

a. Ecrire une fonction *sommesstab* qui prend en paramètre un tableau d'entier tab, et 2 entiers i et j, et qui renvoie un entier valant la somme des cases du tableau tab entre les indices i et j (inclus).
Par exemple, si tab = { -6,12,-7,0,14,-7,5 },
sommesstab(tab, 1, 4) = tab[1] + tab[2] + tab[3] + tab[4] =12-7+0+14 =19

b. Ecrire une fonction *sousTabMax* qui prend en paramètre un tableau d'entiers (positifs, nuls, ou négatifs), et renvoie la somme des éléments du sous tableau qui la rend maximale. On ne s'intéressera ici qu'aux sous tableaux correspondant à des cases *adjacentes* du tableau initial (il ne peut pas y avoir de trou)
Sur tab = { -6,12,-7,0,14,-7,5 }, la fonction doit renvoyer 19 (sous tableau de taille = 4)
Sur tab = { -6,-12,-7,-1,-7,-5 }, la fonction doit renvoyer -1 (sous tableau de taille=1)

Pour aller plus loin : voir l'algorithme de Kadane sur https://en.wikipedia.org/wiki/Maximum_subarray_problem

Exercice 19 Chaines de caractères : fonction rechercher
Trouver toute les occurrences d'une chaine de caractères (motif) dans un texte (une chaine plus longue) est un problème qui se rencontre souvent dans les éditeurs de texte.

Ecrire une fonction *rechercher* qui prend en paramètre 2 chaines de caractère *motif* et *texte,* et qui renvoie un entier valant l'indice (ou le décalage) dans *texte* où *motif* apparait, ou -1 si *motif* n'apparait pas. Par exemple, si *texte* vaut ''un homme averti en vaut deux'' et *motif* vaut ''om'', la fonction doit renvoyer 4

Exercice 20 Chaines de caractères : fonction remplacer
Ecrire une fonction *remplacer* qui prend en paramètre 3 chaines de caractère *motif, patch* et *texte,* et qui renvoie la chaine *texte* où toutes les occurrences de *motif* ont été remplacées par *patch*

Exercice 21 Chaines de caractères : recherche la plus grande sous chaine commune

1. Ecrire une fonction *souchmax* qui prend en paramètre 2 chaines de caractère *adn1 et adn2* , et qui renvoie une chaine qui vaut la plus grande sous chaine communes à *adn1 et adn2.*
Par exemple, souchmax(''crotale'', ''hippopotame'') doit renvoyer ''ota''. S'il existe plusieurs sous chaines communes, renvoyer celle qui apparait en premier dans adn1.
Remarque :
   - adn1 et adn2 pourront être des String, des tableaux de caractères, ou bien même des tableaux d'entiers.
   - "toto".`toCharArray()` transforme le String "toto" en un tableau de char `{'t', 'o', 't', 'o'}`

- Si adn1 et 2 sont de type String, on peut accéder au caractère à l'indice i avec `charAt(i)`

2. Solution rapide

Indice : se baser sur le tableau à double indice suivant (à construire à partir des 2 chaines) :

int[][] grille = new int[''crotale''.length()][''hippopotame''.length()]

	H	I	P	P	O	P	O	T	A	M	E
C											
R											
O					1		1				
T								2			
A									3		
L											
E											1

```
void setup() {
  String adn1 = "crotale";
  String adn2 = "hippopotame";
  int[][] tab = new int[adn1.length()][ adn2.length()];
  printTab(tab, adn1, adn2);
}

void printTab(int[][] tab, String adn1, String adn2) {
  print("   ");
  for (int i = 0; i<adn1.length(); i++) {
    print(adn1.charAt(i), " ");
  }
  println();
  for (int j = 0; j<adn2.length(); j++) {
    print(adn2.charAt(j), " ");
    for (int i = 0; i<adn1.length(); i++) {
      print(tab[i][j], " ");
    }
    println();
  }
}
```

Enumération et structure itérative:

1. *l'énumération de toutes les paires d'éléments d'un tableau se fait avec 2 boucles imbriquées*
2. *l'énumération de tous les trios d'éléments d'un tableau se fait avec 3 boucles imbriquées*
3. *…*

Chapitre 4

Récursivité

Fonctions récursives

Définition : Une fonction récursive est une fonction qui s'appelle elle-même. C'est-à-dire qu'elle contient un appel à elle-même dans sa définition (déclaration)

Une fonction récursive doit comporter :

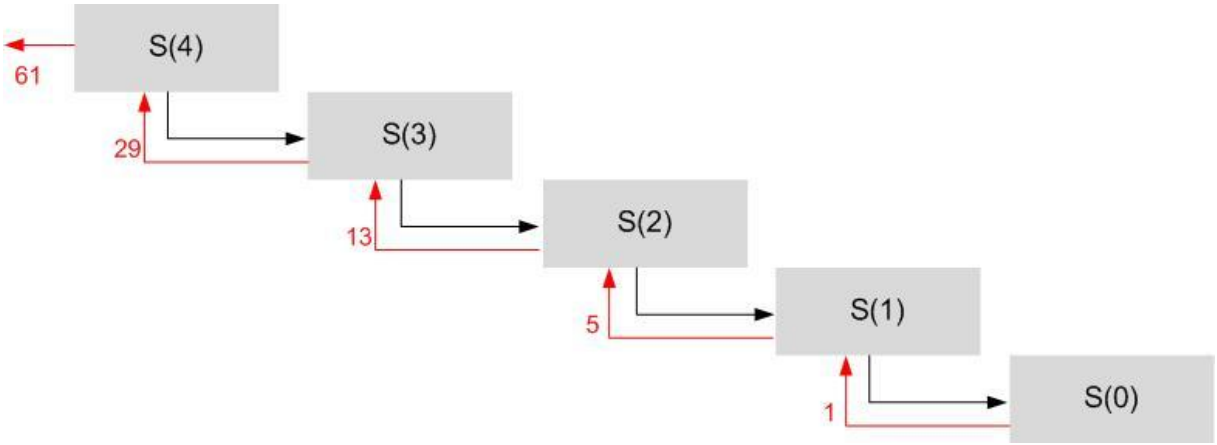
- **un appel récursif (ou plusieurs)** C'est le mécanisme, issu d'une relation de reccurrence, qui détermine l'élément suivant dans la séquence des appels de la fonction.
- **une condition d'arrêt**. Son rôle est de stopper la suite des appels récursifs
- **une relation simple**. C'est ce qui doit être exécuté par la fonction quand la condition d'arrêt est vérifiée.

Pseudo code	Processing

FONCTION s(ENT n) RENVOIE ENT
DEBUT
SI n==0
RENVOIE 1
SINON
RENVOIE 2* s(n-1) +3
FSI
FIN s
	int s(int n) {
	if(s==0)
	return1 ;
	else
	return 2* s(n-1) +3 ;
	}
	
	void setup(){

ALGO test Afficher(s(4)) FIN test	print(s(4)) ; }

Correction :



Exercice 1

Que pensez-vous de la fonction suivante ?

```
FONCTION suspect1(ENT n)
DEBUT
        suspect1 (n+1)
FIN
```

Exercice 1.5

Que pensez-vous du code suivant ?

```
FONCTION suspect2(ENT n)
DEBUT
        si n != 0 alors
                suspect2(n)
FIN
```

Exercice 2 Fonction mystère

```
FONCTION g (ENT n) RENVOIE ENT
DEBUT
        SI n<3
                ALORS RENVOIE 0
        FSI
        RENVOIE g(n-1)-1
FIN g
```

Calculer les valeurs de :

- g(5)
- g(7)
- g(n), pour un n quelconque

Exercice 3 Fonction mystère

Que calcule la fonction f ? Donner une formule en fonction de n équivalente à la fonction f

```
FONCTION f (ENT n,m) RENVOIE ENT
DEBUT
        SI m == 1
                ALORS RENVOIE 1
        FSI
        RENVOIE n*f(n,m-1)
FIN f
```

Calculer les valeurs de :

- f(3,2)
- f(2,5)
- f(1, 101)

Exercice 3.5 Fonction som récursive

Ecrire une fonction som récursive qui prend en parmaètre 2 entiers a et b, supposés positifs, et qui renvoie un entier valant a+b.

Contrainte :

- on pourra écrire a+1, a-1, b+1, b-1 mais pas a+b
- sans boucle

Exercice 4 Factorielle

Ecrire une fonction récursive *factRec* qui étant donné un entier n (supposé positif), renvoie la factorielle de ce nombre : n!

Exercice 4.5

Ecrire une fonction récursive *f* qui étant donné un entier n (supposé positif), renvoie n ! + (n-1) ! + (n-2) ! + … 2 ! + 1 !

Exercice 5 Puissance

Ecrire une fonction récursive *factPuiss* qui étant donné deux entiers x et n (supposé positif) renvoie x à la puissance n

Exercice 5.5 Fonction modulo récursive

Transformez la fonction modulo ci-dessous en une fonction récursive sans boucle

```
int modulo(int a, int b){
      while(a>=b){
          a=a-b
      }
      return a
}
```

Puis faites de même avec la fonction PGCD d'euclide

Exercice 6 Simulation

Qu'est ce qui est affiché suite à l'instruction f(3) ?

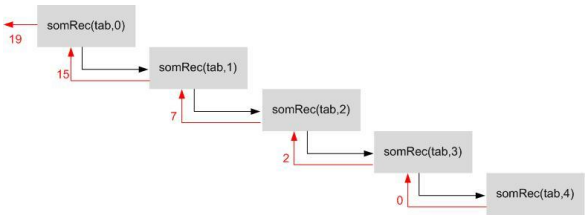
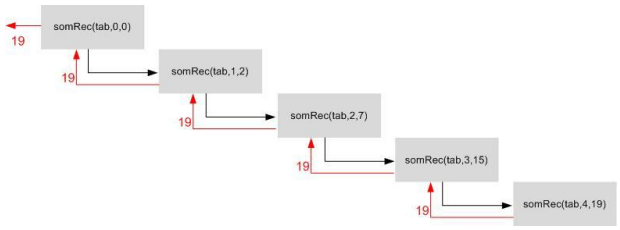
```
FONCTION f (ENT n)
DEBUT
        Afficher (‘’ Appel’’ , n)
        SI n>0
                ALORS Afficher (‘’ Avant’’ , n-1)
                        f(n-1)
                        Afficher (‘’ Après’’ , n-1)

        FSI
FIN f
```

Exercice 7 Itération en récursivité

Contrainte : interdit d'utiliser des structures itératives (POUR ou ITER). Indiquer pour chaque fonction, l'algorithme qui comprend la première instruction d'appel.

a. Ecrire une fonction *IterRec* récursive qui prend en paramètre un entier n et qui produit l'affichage de n fois le message « bonjour ».
b. Ecrire une fonction *parTab* récursive qui prend en paramètre un tableau d'entiers et qui en affiche les valeurs de gauche à droite. Comment modifier l'ordre d'affichage des éléments sans changer le premier appel de la fonction *parTab ?*

c. Ecrire une fonction *SomRec* récursive qui prend en paramètre un tableau d'entiers et qui renvoie la somme de ses éléments.

Correction pour la fonction somRec	
Fonction somRec(ent[] tab, ent k) renvoie ENT Si k==taille(tab) alors renvoie 0 fsi renvoie tab[k] + somRec(tab, k+1)	Fonction somRec(ent[] tab, ent k, ent som) renvoie ENT Si k==taille(tab) alors renvoie som fsi renvoie somRec(tab, k+1, som+tab[k])
1er appel : k=0	**1er appel : k=0 et som=0**
Execution sur tab = {2, 5, 8, 4}	
	

Exercice 9 Fibonacci

Ecrire une fonction récursive *fiboRec* qui étant donné un entier n permet de calculer le n-ième terme de la suite de Fibonacci définie par :

$$U_n = U_{n-1} + U_{n-2}\ \forall n > 1 \text{ avec } U_0 = 0 \text{ et } U_1 = 1$$

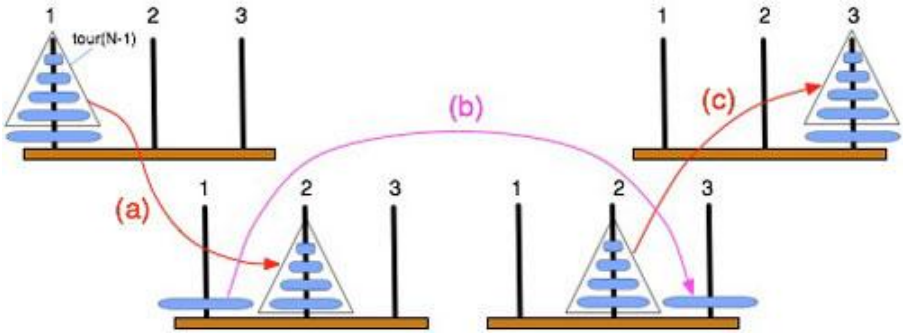
Exercice 11 Le jeu des Tours de Hanoï.

Une tour de Hanoï de hauteur N, appelée tour(N) dans la suite, est constituée d'un ensemble de N **d**isques de diamètres différents, percés d'un trou central, et enfilés sur un plot vertical par ordre de diamètre décroissant. On dispose de trois plots appelés 1, 2, 3. Une tour(N) est initialement installée sur le plot 1. Le problème consiste à la déplacer vers le plot 3, en utilisant le plot 2 comme lieu de rangement intermédiaire et en respectant les règles suivantes :

- on ne peut déplacer qu'un disque à la fois ;
- on ne peut déplacer qu'un disque situé au sommet de sa pile ;
- on ne peut poser un disque que sur un disque de diamètre supérieur ou sur un plot vide.

Notons que le problème pour une tour(N) peut être résolu simplement si on sait le résoudre pour une tour(N-1). En effet une tour(N) est un disque unique surmonté par une tour(N-1). On transporte alors cette tour(N-1) du plot 1 au plot 2 avec le plot 3 comme intermédiaire (opération a). On déplace le disque restant du plot 1 au plot 3 (opération b). Enfin, on transporte la tour(N-1) du plot 2 au plot

3 avec le plot 1 comme intermédiaire (opération c), ce qui termine le travail. La figure ci-dessous schématise ces opérations.



On ramène ainsi le problème pour N disques au problème pour N-1 disques, et on arrive ainsi de proche en proche à un disque unique, pour lequel la solution est immédiate.

1. Ecrire une fonction *deplace(E Ent n, Ent i, j)* qui affiche les déplacements de n disques du plot i au plot j. Par exemple, *deplace(2,1,3)* doit afficher les déplacements de 2 disques du plot 1 au plot 3. A chaque déplacement de disque on affiche uniquement le plot d'origine et le plot d'arrivée :

   1→2,
   1→3,
   2→3

   Indice : si i et j sont des valeurs distinctes de l'ensemble {1, 2, 3}, le numéro du dernier plot k vaut 6 – i – j

2. Vérifiez que le nombre de mouvements pour déplacer n disques vaut $2^n - 1$

Exercice 12 Fonction de Mc Carthy

Soit f une fonction définie sur l'ensemble $\mathbb{N}$ comme :

$$f(n) = n - 10 \; si \; n > 100$$

$$= f\big(f(n+11)\big) \quad sinon$$

1. Ecrire la fonction sous une forme récursive
2. Donner une formule en fonction de n pour le calcul de f quand n<101

Exercice 12.5 Chaine de bits de taille n

Exercice 1 : affichages des chaines de bits de taille n

Par exemple, pour n = 2 , la console doit afficher : 00, 01, 10, 11

Soit tab un tableau de taille n. On cherche à remplir ce tableau de toutes les manières possibles avec des 0 et des 1. Pour ce faire, on va écrire une fonction récursive void chainesBits(int [] tab, int ind) qui devra :

- affecter une valeur à tab[ind] (0 ou 1)
- s'appeler sur l'indice d'après

Début de solution à compléter :

```
void chainesBits ( int[] tab,  int ind) :
    if ???
    else :
        tab[ind]=0
        chainesBits (tab, ind+1)
        tab[ind]=1
        chainesBits (tab, ind+1)

void printChainesBits(int n):
    int[] tab= new int[n]
    chainesBits(tab,0)
```

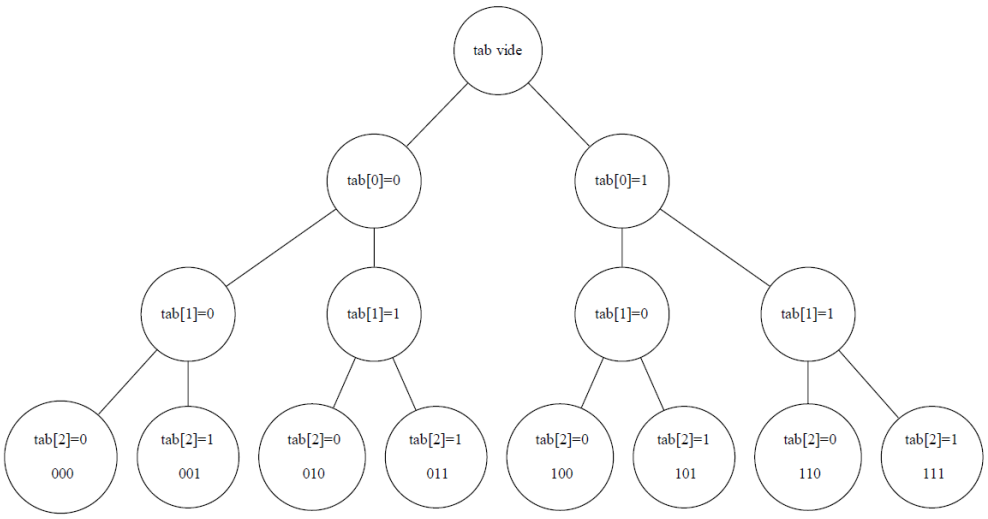
Correction :

	`void chainesBits ( int[]tab,  int ind) :`
C	`    if (ind == len(tab))println(tab)` `    else :`
A	`    tab[ind]=0` `    chainesBits (tab, ind+1)`
B	`    tab[ind]=1` `    chainesBits (tab, ind+1)`

Notation (abreviation) : chainesBits (tab, ind) = cb(ind)

cb(0)				Console
A : tab[0]=0	cb(1)			
	A : tab[1]=0	cb(2)		
		A : tab[2]=0	cb(3)	
			C →	000
		B : tab[2]=1	cb(3)	
			C →	001
	B : tab[1]=1	cb(2)		
		A : tab[2]=0	cb(3)	

			C →	010
		B : tab[2]=1	cb(3)	
			C →	011
B : tab[0]=1	cb(1)			
	A : tab[1]=0	cb(2)		
		A : tab[2]=0	cb(3)	
			C →	100
		B : tab[2]=1	cb(3)	
			C →	101
	B : tab[1]=1	cb(2)		
		A : tab[2]=0	cb(3)	
			C →	110
		B : tab[2]=1	cb(3)	
			C →	111



Exercice 13 Enumération des n-uplets -

Dans cet exercice, on regarde comment construire des algorithmes basés sur l'exploration de l'espace des solutions possibles.

Par exemple, pour savoir si un tableau contient des doublons, on énumère l'espace de tous les couples d'indices (avec 2 boucles imbriquées) et on applique le traitement à l'énumération de chaque nouveau couple d'indices (vérifier si les 2 cases correspondantes dans le tableau ont même valeur). Si E désigne la taille du tableau, on dit que la solution est formée :

- d'une structure itérative permettant l'énumération de l'ensemble des couples d'éléments de E (= les éléments de E^2)
- d'un traitement appliqué à chaque énumération

Dans la suite, on focalise sur la construction des structures itératives, le traitement se réduira à l'affichage de l'élément énuméré.

a. Qu'est ce qui est affiché par l'algorithme enumere3uplet ?

```
ALGO enumere3uplet
        Tableau tab ENT[3]
        ENT i, j, k

        POUR (i=0, i< 8; i++)
                POUR (j=0, j< 8; j++)
                        POUR (k=0, k< 8; k++)
                                tab[0]=i
                                tab[1]=j
                                tab[2]=k
                                Afficher Tab(tab)
                                Afficher(alaligne)
                        FPOUR
                FPOUR
        FPOUR
FIN enumere3uplet
```

```
FONCTION AfficherTab (E ENT[] tab)
        Ent i
        POUR (i=0, i<taille(tab), i++)
                Afficher(tab[i])
        FPOUR
FIN AfficherTab
```

Solution :

L'algorithme affiche toutes les combinaisons possibles de 3 nombres de l'ensemble E = {0, 1, …, 7}, où n est un entier saisi par l'utilisateur. On dit que ce sont les 3-uplets de E qui sont énumérés (un 2-uplet est un couple et un 3-uplet est un triplet ; un 4-uplet est un quadruplet).

Remarques :

a. La plage des valeurs (de 0 à 7) est ici arbitraire. On aurait tout aussi bien pu la demander à l'utilisateur.
b. On aurait pu placer l'instruction tab[0]=i entre les lignes POUR (i=0, i< 8; i++) et POUR (j=0, j< 8; j++) sans changer le résultat. Ceci est aussi valable pour l'instruction tab[1]=j

En termes d'exploration d'un espace de recherche, les fonctions itératives fournissent un bon cadre quand la dimension de l'espace est connue au moment de l'écriture du code : il suffit d'imbriquer autant de boucles, et on peut parcourir tout l'espace de recherche au milieu de la boucle la plus centrale. Mais quand cette dimension est un paramètre qui n'est connu qu'à l'exécution (un entier saisi par l'utilisateur par exemple), cette approche ne fonctionne plus. Ceci est en revanche possible avec des fonctions récursives. Dans la fonction ci-dessous, on montre comment parcourir l'espace E^n , où n et E sont définis à l'exécution.

b. Simulez l'exécution de la fonction *enum* puis corrigez-la pour qu'elle fonctionne correctement

```
ALGO enumereNuplet
        Ent n, cardE

        Afficher(''entrer n '')
        Saisir(n)
        Tableau tab ENT[n]
        enum(tab,0)
Fin enumereNuplet

FONCTION enum ( ENT[] tab, E ENT n)
```

```
        Ent val
        SI n == taille (tab)
                ALORS  AfficherTab(tab)
                        Afficher (alaligne)
        FSI

        POUR (val=0, val< 8 ; val++)
                tab[n]=val
                enum(tab, n+1)
        FPOUR
FIN enumere3uplet
```

c. Application 1 : Sous ensemble de somme nulle

- Ecrire une fonction qui prend en paramètre un tableau d'entiers, et qui indique si la somme de certains (en nombre quelconque) de ses coefficients vaut 0. Par exemple, sur le tableau tab = {4 , 6, 5, -3, 1, -12, 9, 400} la fonction doit afficher (textuellement) « 6+(-3)+9+(-12) = 0 ». S'il n'existe aucune solution, elle affichera « aucune solution »
- Modifier votre fonction pour qu'elle affiche toutes les solutions s'il en existe plusieurs

d. Application 2 : Problème des n-reines

Le but dans le problème des n reines est de placer n dames d'un jeu d'échecs sur un échiquier de n×n cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs (la couleur des pièces étant ignorée). Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale. L'objectif est de trouver une solution en fonction d'un entier n fourni par l'utilisateur.

On propose coder une configuration des n reines sur l'échiquier à l'aide d'un tableau mono indice. Par exemple, pour n = 4, une des solutions est la configuration suivante :

		x	
x			
			x
	x		

Codage par numérotation des reines dans chaque colonne (vers le bas en partant de 0) :

1	3	0	2

- version 1 : en modifiant la fonction enum de manière à parcourir tous les tableaux de n entiers strictement inférieurs à n.

- version 2 : **backtracking** : en incorporant un mécanisme de retour sur trace qui consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. Par exemple, ceci permet de ne pas continuer à « avancer » si le début du tableau est

1	1		

  - On pourra définir une fonction *inserable*(ENT[] tab, ENT val,ENT r) qui renvoie un booléen valant VRAI si la valeur val est insérable dans le tableau tab au rang r, et FAUX sinon. Par exemple, inserable(tab, 1, 1) doit renvoyer FAUX si tab ={1,-,-,-}. («-» désigne des valeurs pas encore attribuées. On peut le remplacer par -1)
  - On pourra définir une fonction resol(ENT[] tab, ENT rang) qui renvoie un booléen valant VRAI si il existe une solution, et FAUX sinon
  - Modifier votre algo de manière à lister toutes les configurations gagnantes pour un n donné

Pseudo code	Processing	C
ent		int
reel		float
bool	boolean	Ø
car		char
chaine	String	char[]

```
var = expr
```

saisir(var)	Ø	scanf(&var)

```
SI expr_log                          if (expr_log){
     ALORS                                instruction1
          instruction1                    instruction2
          instruction2               } else {
     SINON                                instruction3
          instruction3                    instruction4
          instruction4               }
FSI
```

afficher(expression)	print(expression)	printf("%d", expression) si expression est de type int

```
ITER                                 while (expr_log0){
     SORTIRSI expr_log0                   instruction1
     instruction1                         instruction2
     SORTIRSI expr_log1                   instruction3
     instruction2                         instruction4
     instruction3                    }
     SORTIRSI expr_log2
```

```
      instruction4                          do{
      SORTIRSI expr_log3                          instruction1
FITER                                             instruction2
                                                  instruction3
                                                  instruction4
                                            }while(expr_log3) ;


POUR(instr_init;expr_log;instr)   for(instr_init; expr_log;instr){
      instruction1                      instruction1
      instruction2                      instruction2
      instruction3                      instruction3
      instruction4                      instruction4
FPOUR                             }
```

Petit mémento des correspondances syntaxiques

entre langages de programmation de 1ère année

ESGI 1i

Domaine	Pattern	Langage C	Processing	Pseudo Code	PHP	JavaScript
Itérations	Boucle for pour 5 itérations	`for(int i=0 ;i<5 ;i++)`	`for(int i=0 ;i<5 ;i++)`	`Ent i POUR (i=0 ; i <5 ; i++)`	`for ($i = 1; $i < 5; $i++)`	`for (let i = 0; i < 5; i++)`
Affichage sur console	Afficher la valeur d'une variable x de type entier	`printf("%d",x);`	`print(x) ;`	`AFFICHER(x)`	`echo $x;`	`console.log(x);`
Tableaux	Déclarer un tableau mono indice	Tableau d'entiers à taille fixe			Tableau à taille variable	
		`int tab[n] ;`	`int[] tab = new int[n] ;`	`Tableau ENT[n]`	`$tab = [];`	`const tab=[] ;`
	Taille d'un tableau	`sizeof(tab)/ sizeof(tab[0]) (cas statique)`	`tab.length`	`taille(tab)`	`count($tab)`	`tab.length`
Fonctions	Déclarer une fonction	`int moy(int a, int b){ return (a+b)/2 ; }`	`int moy(int a, int b){ return (a+b)/2 ; }`	`Fonction moy(Ent a, Ent b) renvoie Ent renvoie (a+b)/2 ;`	`function moy($a, $b){ return ($a+$b)/2 ; }`	`function moy (a, b) { return (a+b)/2 ; }`
	Appel	`moy(4, 6)`	`moy(4, 6)`	`moy(4, 6)`	`moy(4, 6)`	`moy(4, 6)`