[X1100101]

Introduction à l'informatique

Exemplaire Enseignant

Livret d'exercices

NB: corrections indicatives et non garanties....

UNIVERSITÉ DE NANTES

Table des matières

Initiation à l'Algorithmique

RAPPEL DE COURS

DÉFINITION

Un algorithme est la description non ambiguë dans un langage convenu d'une méthode de résolution d'un problème.
Il est généralement destiné à être exécuté par une autre personne que celle qui l'a écrit, ou à être traduit dans un langage informatique pour être exécuté par une machine. Il peut aussi être objet d'étude (vitesse, comparaison à d'autres, ...), ou créé automatiquement par un ordinateur.

ÉCRITURE D'UN ALGORITHME

Un algorithme comprend un entête, une partie déclaration, une partie actions et des commentaires.
L'entête comprend le nom de l'algorithme et un commentaire sur sa fonction.
La déclaration précise pour chaque variable utilisée son nom (identificateur), son type et son rôle.
La partie actions décrit dans l'ordre les actions à exécuter pour résoudre le problème.
Les commentaires ne sont pas destinés à l'exécutant, ils facilitent la lecture et la modification de l'algorithme par l'analyste.

ACTIONS ÉLÉMENTAIRES

Saisie :	`<nom de variable>` ← **`Saisie()`**	*met dans la variable la valeur tapée au clavier*
Affectation :	`<nom de variable>` ← `<expression>`	*met dans la variable la valeur de l'expression*
Affichage :	**`Ecrire(`**`<expression>`**`)`**	*envoie à l'écran la valeur de l'expression*

VARIABLE

Une variable possède un identificateur (nom), un type et une valeur (qui peut changer au fil de l'exécution). Les types possibles sont :

Réel *:*	*comme en mathématique, souvent représenté en notation scientifique : mantisse E exposant*
Entier *:*	*entier relatif*
Caractère *:*	*exactement un symbole (lettre, chiffre, symbole d'opération, ponctuation, espace, perluette, ...)*
Chaîne de caractères *:*	*suite finie (éventuellement vide) de caractères*
Booléen *:*	*élément de l'ensemble { vrai , faux } aussi notés V / F ou true / false.*

EXPRESSION

Une expression possède une écriture, un type et une valeur — qui peut dépendre du contenu de variable(s). Le type dépend de l'écriture de l'expression, pas de sa valeur ; il est calculé à partir des types des opérandes et des opérateurs.

HISTORIQUE D'EXÉCUTION

On exécute normalement un algorithme en suivant l'ordre dans lequel les actions sont écrites (exécution séquentielle). Pour garder une trace de ce qu'ont contenu les variables pendant l'exécution, on trace un tableau avec une colonne pour chaque variable présente dans l'algorithme, en remplissant une ligne après exécution de chaque action. On numérote les actions dans l'algorithme et en conséquence dans l'historique. Chaque ligne représente l'état des variables à un moment donné de l'exécution.

Expression

Exercice A.1

Donner le type (ou les types possibles) puis (éventuellement) la valeur des expressions suivantes, dans lesquelles le symbole ␣ représente un espace :

Expression	Type (ou types possibles)	Valeur (si évaluable)
3*7	Numérique	21
3+7	Numérique	10
'3'+'7'	Chaîne de caractères	'37'
0	Numérique	0
'0'	Chaîne de caractères	'0'
un	Ceci est une variable	valeur stockée
'un'	Chaîne de caractères	'un'
'un' + 'un'	Chaîne de caractères	'unun'
3.7	Numérique réel	3.7
'3.7'	Chaîne de caractères	'un'
17/3	Numérique réel	5.66666667
17 div 3	Numérique entier	5
17 mod 3	Numérique entier	2
'2' + '␣' + '3'	Chaîne de caractères	'2 3'

Exercice A.2

Donner le type (ou les types possibles) puis (éventuellement) la valeur des expressions maladroites suivantes, dans lesquelles le symbole ␣ représente un espace, en explicitant les conversions de type à effectuer, et les types que peuvent ou doivent avoir les variables :

Lorsque l'expression possède des types qui ne correspondent pas aux types requis par les opérateurs, il y a une phase de conversion de type que l'on peut réaliser « manuellement » à l'aide des trois fonctions enChaine, enEntier et enReel. Il faut tout de même noter que certains signes d'opération (ex: +, <, =,…), plusieurs solutions de conversion sont envisageables ('3'+7 pourrait aussi bien être l'addition que la concaténation). Dans tous les cas où à la fois la conversion en chaîne d'un des opérandes et la conversion en nombre de l'autre sont possibles, c'est la conversion de chaîne qui est préférée avec comme argument que tout nombre peut se transformer en chaîne tandis que toute chaîne ne peut pas se transformer en nombre…

RAPPEL DE COURS

STRUCTURE CONDITIONNELLE

On exécute normalement un algorithme dans l'ordre dans lequel les actions sont écrites (exécution séquentielle). Il est cependant parfois nécessaire de pouvoir choisir **lors de l'exécution** entre deux « chemins ». On utilise pour cela une structure conditionnelle.

ÉCRITURE

si (*expression booléenne*) **Alors** <*actions si vrai*> **Sinon** <*actions si faux*> **fin si**

EXÉCUTION

L'expression booléenne est évaluée, le résultat est soit **vrai**, soit **faux**. Si le résultat est **vrai**, les actions entre **alors** et **sinon** sont exécutées (mais pas les autres) ; si en revanche le résultat est **faux**, ce sont les actions entre **sinon** et **fin si** qui sont exécutées. Tout ce qui précède le **si** est exécuté normalement, ainsi que tout ce qui suit le **fin si**.

REMARQUES

– La suite d'instructions derrière le **sinon** peut être vide ; on omet alors le **sinon** et on parle de conditionnelle simple (par opposition à alternative).

– L'<*expression booléenne*> peut être une comparaison (**si** (x<3) **Alors**...), une expression booléenne complexe (**si** (x≠y ou z<7) **Alors**...), ou une variable booléenne (**si** (trouvé) **Alors**...).

Expression	Type (ou type possibles de l'expression)	Valeur (si évaluable)	Expression correcte, avec conversion(s) et types des variables
'3'*7	Numérique	21	enEntier('3')*7
'3'+7	Chaîne de caractères	'37'	'3'+enChaine(7)
'3.7'+2	Chaîne de caractères	'3.72'	'3.7'+enChaine(2)
'3'*1+2	Numérique	5	enEntier('3')*1+2
'3'*(1+2)	Numérique	9	enEntier('3')*(1+2)
('5'+3)*('2'+1)	Numérique	1113	enEntier('5'+enChaine(3))*enEntier('2'+enChaine(1)) Il faut insister sur l'ordre d'évaluation des opérations
('5'+3)*('2'-1)	Numérique	53	enEntier('5'+enChaine(3))*(enEntier('2')-1)
2+'␣'+3	Chaîne de caractères	'2 3'	enChaine(2)+' '+enChaine(3)
3<12	Booléen	VRAI	
40≠15	Booléen	VRAI	
6≠3*2	Booléen	FAUX	
x*12.32	Numérique	dépend de x	enReel(x)*12.32
y*z	Numérique	dépend de y et z	enReel(y)*enReel(z)
y+z	Inconnu	dépend de y et z	

Expression	Type (ou type possibles de l'expression)	Valeur (si évaluable)	Expression correcte, avec conversion(s) et types des variables
`x+'1'`	Chaîne de caractères	dépend de x	enChaine(x)+'1'

Séquentialité

Exercice A.3

On considère la suite d'instructions ci-contre. Remplir le tableau ci-dessous présentant sur chaque ligne l'état des variables à l'issue de l'exécution de chaque instruction.

```
demi ← 6 ;
double ← demi * 2 ;
demi ←   demi / 2 ;
double ← demi + demi ;
```

Instructions	demi	double
Avant exécution	?	?
Après `demi ← 6`	6	?
Après `double ← demi * 2`	6	12
Après `demi ← demi / 2`	3	12
Après `double ← demi + demi`	3	6

Exercice A.4

```
1    top ← 6;
2    bot ← 2;
3    top ← top - bot;
4    bot ← bot + bot;
5    bot ← bot + top;
```

- Établir comme dans l'exercice 3 un historique d'exécution pour la suite d'instructions ci- contre. Pour plus de commodité, les instructions ont été numérotées de 1 à 5.
- Établir un nouvel historique pour la même suite d'instructions dans laquelle on aurait permuté les instructions 3 et 4 .
- Trouver un ordre de ces instructions pour lequel la variable `bot` aurait comme valeur finale 16

Instructions	top	bot
Avant 1	?	?
Après 1	6	?
Après 2	6	2
Après 3	4	2
Après 4	4	4
Après 5	4	8

Instructions	top	bot
Avant 1	?	?
Après 1	6	?
Après 2	6	2
Après 4	6	4
Après 3	2	4
Après 5	2	6

Deux ordres sont possibles: 1,2,5,4,3 et 1,2,5,3,4.

Exercice A.5

Écrire la suite d'actions affectant à une variable l'année de naissance d'une personne puis calculant et affichant son âge au 31 décembre 2010.

Algorithme Calcule age
Variables:
    age, année: entiers;
Début
    année ← Saisie();
    age ← 2010 - année;
    Ecrire(age);
Fin

Exercice A.6

Écrire la suite d'affectations calculant puis affichant la distance `d` à vol d'oiseau entre deux villes situées sur l'équateur. Les longitudes en degrés de ces deux villes sont d'abord stockées dans deux variables `long1` et `long2`. On rappelle que la terre à un diamètre de 12756 km. On ne traitera que le cas simple (différence des longitudes <180).

NB: on s'agit de calculer la longueur d'un arc de cercle (de diamètre 12756) dont l'angle est |long2-long1| (donné en degrés qu'il faut convertir).

Algorithme Distance
Variables:
    long1, long2: entiers;
    distance: réel;
Début
    long1 ← Saisie();
    long2 ← Saisie();
    distance ← |long2-long1|*3.14/180*12756/2;
    Ecrire(distance);
Fin

Exercice A.7

La suite d'actions ci-contre affiche les valeurs données par l'utilisateur. Remplacer la 3ème instruction (qui ne fait rien d'intelligent) par une ou plusieurs instructions permutant le contenu des variables `Aba` et `Coc` de façon à ce que la première valeur affichée soit la seconde donnée, et la seconde affichée, la première donnée.

```
Aba ← Saisie() ;
Coc ← Saisie() ;
Aba ← Aba ;
Ecrire(Aba) ;
Ecrire(Coc) ;
```

NB: il s'agit ici de montrer comment on doit programmer l'échange de deux variables, en utilisant une variable temporaire.

tmp ← Aba;
Aba ← Coc;
Coc ← tmp;

Il est aussi possible de dresser l'historique d'exécution pour bien expliquer comment cela fonctionne.

A noter aussi, dans le cas de nombre, il y a aussi la solution de le faire par les trois instructions

Aba ← Aba + Coc; Coc ← Aba - Coc; Aba ← Aba - Coc;

Cette solution, qui n'utilise pas de variable supplémentaire, est parfois trouvée par quelques étudiants….

Exercice A.8

Écrire un algorithme demandant à l'utilisateur le prix unitaire hors taxe d'un article ainsi que la quantité désirée et affichant le prix total TTC de la commande.

Algorithme Prix
Variables:
    quantité: entier;
    prixHT, prixTotal: réels;
Début
    prixHT ← Saisie();
    quantité ← Saisie();
    prixTotal ← prixHT*1.206*quantité;
    Ecrire(prixTotal);
Fin

Exercice A.9

On désire imprimer des images prises par un appareil photo numérique. Ces images sont des rectangles de 1600 pixels de large et de 1200 pixels de haut. On peut imprimer ces images à différentes résolutions (typiquement 150 pixels par pouce mais des valeurs de 100 à 1200 sont possibles ; en dessous de 100 ppp, des effets d'escalier sont visibles à l'oeil nu). On se demande quelle sera la taille (en cm) des images une fois imprimées. NB: un pouce vaut environ 2,54 cm. Ecrire l'algorithme permettant de calculer cette taille.

Algorithme Photos
Variables:
    largeurPX, hauteurPX,résolution: entiers;
    largeurCM, hauteurCM: reels;
Début
    résolution ← Saisie();
    largeurPX ← 1600;

```
    hauteurPX ← 1200;
    largeurCM ← largeurPX / résolution * 2.54;
    hauteurCM ← hauteurPX / résolution * 2.54;
    Ecrire(largeurCM);
    Ecrire(hauteurCM);
Fin
```

Conditionnelle unique

Exercice A.10

Écrire un algorithme demandant l'année de naissance d'une personne et affichant son âge au 31 décembre 2010 à minuit. Une personne née en 2010 ou plus tard doit avoir 0 an.

```
Algorithme Calcule age
Variables:
    age, année: entiers;
Début
    année ← Saisie();
    age ← 2010 - année;
    si (age < 0)
        Alors age ← 0;
    fin si
    Ecrire(age);
Fin
```

Exercice A.11

Écrire un algorithme demandant deux mots (dans deux chaînes de caractères) et affichant le premier de ces deux mots dans l'ordre alphabétique. Remarque : l'opérateur de comparaison < signifie justement l'ordre alphabétique lorsqu'il est appliqué à des chaînes de caractères (avec quelques restrictions :-).

```
Algorithme le premier
Variables:
    mot1, mot2, premier: chaînes de caractères;
Début
    mot1 ← Saisie();
    mot2 ← Saisie();
    si (mot1 < mot2)
        Alors premier ← mot1;
        Sinon premier ← mot2;
    fin si
    Ecrire(premier);
Fin
```

Conditionnelles imbriquées

Exercice A.12

Écrire un algorithme demandant à l'utilisateur une année du calendrier grégorien puis calculant un booléen exprimant si ce millésime est celui d'une année bissextile. L'algorithme doit ensuite afficher un message indiquant si cette année est bissextile.
On rappelle qu'une année est bissextile si (et seulement si) son millésime est : -- soit divisible par 4 mais pas par 100 -- soit divisible par 400 Envisager une écriture avec des conditions simples, et une autre avec une condition complexe.
Pour cet exercice, il est utile de dessiner un arbre de décision et d'expliquer comment on peut passer d'un arbre de décision à du code algorithmique.

```
Algorithme bissextile
Variables:
    année: entier;
    bissextile: booléen;
Début
    année ← Saisie();
    // Avec des conditions simples
    si (année mod 4 = 0)
        Alors si (année mod 100 = 0)
                Alors si (année mod 400 = 0)
                        Alors bissextile ← Vrai;
                        Sinon bissextile ← Faux;
                    fin si
                Sinon bissextile ← Vrai;
            fin si
        Sinon bissextile ← Faux;
    fin si
    // Avec une condition complexe
    bissextile ← ((année mod 4 = 0) et (année mod 100 ≠ 0)) ou (année mod 400 = 0);
    Ecrire(bissextile);
Fin
```

Exercice A.13

Écrire un algorithme demandant le mois et l'année de naissance d'une personne puis calculant et affichant son âge au 31 août 2010 minuit.
Écrire de même un algorithme affichant l'âge d'une personne le 14 juillet 2010 à partir de sa date de naissance.
Discuter préalablement des cas à envisager. Bien commenter.
Pour cet exercice, il est utile de représenter les différents cas où il faut « corriger » l'âge calculé par la formule 2010 - année.

```
Algorithme Calcule âge au 31 août 2010
Variables:
```

```
        mois, année, age: entiers;
Début
        mois ← Saisie();
        année ← Saisie();
        age ← 2010 - année;
        si (age > 0)
                Alors si (mois > 8)
                        Alors age ← age - 1;
                fin si
                Sinon age ← 0;
        fin si
        Ecrire(age);
Fin
```

```
Algorithme Calcule âge au 14 juillet 2010
Variables:
        jour, mois, année, age: entiers;
Début
        jour ← Saisie();
        mois ← Saisie();
        année ← Saisie();
        age ← 2010 - année;
        si (age > 0)
                Alors si ((mois > 7) ou ((mois = 7) et (jour > 14)))
                        // Possibilité d'ajouter une imbrication au « si »
                        Alors age ← age - 1;
                fin si
                Sinon age ← 0;
        fin si
        Ecrire(age);
Fin
```

Exercice A.14

En utilisant l'exercice 12 (identification d'une année bissextile), écrire un algorithme calculant et affichant le nombre de jours dans un mois donné. Les informations demandées à l'utilisateur sont le numéro du mois (1 à 12) et l'année (sur 4 chiffres).
La correction de cet exercice est longue, fastidieuse mais évidente. Pas de correction dans ce livret.

Exercice 15

Écrire un algorithme qui demande la saisie de 3 nombres et qui les affiche dans l'ordre croissant.

Pour cet exercice, la solution demandée consiste à tester tous les 6 cas possibles. Il n'est cependant pas interdit de parler de tri bulle et de donner une version correspondant à un tri bulle de trois valeurs en intérrogeant les étudiants sur ce qui se passerait si on cherchait à faire le même travail pour 4 valeurs.

```
Algorithme ordonne 3 valeurs
Variables:
        a,b,c: entiers;
Début
        a ← Saisie();
        b ← Saisie();
        c ← Saisie();
        si (a < b)
                Alors si (b < c)
                        Alors Ecrire(a+'<'+b+'<'+c);
                        Sinon si (a < c)
                                        Alors Ecrire(a+'<'+c+'≤'+b);
                                        Sinon Ecrire(c+'≤'+a+'<'+b);
                                fin si
                        fin si
                Sinon si (a < c)
                        Alors Ecrire(b+'≤'+a+'<'+c);
                        Sinon si (b < c)
                                        Alors Ecrire(b+'<'+c+'≤'+a);
                                        Sinon Ecrire(c+'≤'+b+'≤'+a);
                                fin si
                        fin si
        fin si
Fin

Algorithme ordonne 3 valeurs (tri bulle)
Variables:
        a,b,c,tmp: entiers;
Début
        a ← Saisie();
        b ← Saisie();
        c ← Saisie();
        si (a < b)
                Alors tmp ← a; a ← b; b ← tmp;
        fin si
        si (b < c)
                Alors tmp ← b; b ← c; c ← tmp;
        fin si
        si (a < b)
                Alors tmp ← a; a ← b; b ← tmp;
        fin si
        Ecrire(a+'≤'+b+'≤'+c);
```

Fin

RAPPEL DE COURS

RÉPÉTITIVE
On exécute normalement un algorithme dans l'ordre dans lequel les actions sont écrites (exécution séquentielle). Il est cependant parfois nécessaire de pouvoir choisir **lors de l'exécution** de refaire une portion de l'algorithme. On utilise pour cela une structure répétitive (familièrement appelée boucle). Les trois types de boucles vues dans ce cours sont:

Tant que

```
Tant que (expression booléenne) faire <actions à répéter> fin tant que
```

Pour

```
Pour i allant de 1 à 10 faire <actions à répéter> fin pour
```

UTILISATIONS DES BOUCLES
Les utilisations des boucles sont très diverses. Cependant, la plupart des boucles à écrire reposent sur trois schémas de programmation très classiques: le compteur; l'accumulateur; et la vérification de saisie. Il est important de savoir reconnaître ces schémas et de savoir les écrire «par coeur».

Répétitives simples

Exercice 16

Lors des essais pour un grand prix de formule 1, on chronomètre (plusieurs fois) le temps mis par une voiture pour effectuer un tour du circuit. Écrire un algorithme qui demande à l'utilisateur chacun des temps (en nombre entier de secondes pour simplifier) puis affiche le temps total (utiliser un cumul) et le nombre de tours effectués. Dans chacun des cas, vous préciserez le schéma de programmation se rapprochant le plus du travail demandé. Chaque cas correspond plus ou moins à une situation (essais libres, essais qualificatifs et conditions de course)

- Écrire une première version en supposant que l'utilisateur/« entraîneur » donne un temps nul pour arrêter l'algorithme.
- Écrire une seconde version en supposant que l'utilisateur donne un temps total (ex: la durée des essais qualificatifs).
- Écrire une troisième version où on demande à l'utilisateur au préalable le nombre de tours (ajouter un compteur). L'algorithme doit s'arrêter au bout du nombre de tours prévu.

Pour cet exercice, la solution demandée consiste à tester les deux types de boucles ainsi que les trois schémas de programmation (vérification de saisie, accumulateur et compteur).

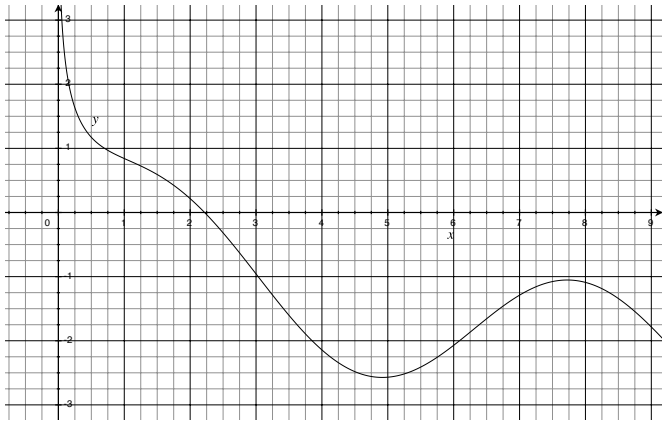
```
Algorithme Formule 1
Variables:
    nbtours, temps, cumul: entiers;
Début
    nbtours ← 0;
    cumul ← 0;
    temps ← Saisie();
    Tant que (temps ≠ 0) faire // Vérification d'une saisie
        nbtours ← nbtours + 1; // Ceci est un compteur
```

```
            cumul ← cumul + temps; // Ceci est un accumulateur
            temps ← Saisie();
        fin tant que
        Ecrire(nbtours);
        Ecrire(cumul);
Fin

Algorithme Formule 2
Variables:
        nbtours, temps, cumul, tempstotal: entiers;
Début
        tempstotal ← Saisie();
        nbtours ← 0;
        cumul ← 0;
        Tant que (cumul < tempstotal) faire
            temps ← Saisie();
            nbtours ← nbtours + 1;
            cumul ← cumul + temps;
        fin tant que
        Ecrire(nbtours);
        Ecrire(cumul);
Fin

Algorithme Formule 3
Variables:
        i,nbtours, temps, cumul: entiers;
Début
        nbtours ← Saisie();
        Pour i allant de 1 à nbtours faire
            temps ← Saisie();
            cumul ← cumul + temps;
        fin pour
        Ecrire(nbtours);
        Ecrire(cumul);
Fin
```

Exercice 17



On cherche à évaluer numériquement (et approximativement) un minimum de la fonction f : x ↦ sin(x)-ln(x) représentée ci-contre. Pour cela, on va calculer la valeur de f(x) pour x variant de 0,1 en 0,1 à partir de 0,2. Au début, les valeurs sont de plus en plus petites, mais lorsque l'on dépasse l'abscisse d'un minimum, les valeurs remontent. Écrire un algorithme qui calcule successivement les valeurs de f(0,2), f(0,3), f(0,4), et ainsi de suite jusqu'à ce qu'une valeur soit plus grande que la précédente. L'algorithme doit alors afficher l'abscisse x correspondante.
La valeur initiale 0,2 et le pas 0,1 seront demandés à l'utilisateur.

Dans cet exercice, on a l'occasion de donner la définition d'une fonction simple (f(x)).

```
Fonction f(x: réel) : réel
Début
    retourner sin(x)-ln(x);
Fin
```

```
Algorithme Maximum fonction
Variables:
    x,pas: réel;
Début
    x ← Saisie();
    pas ← Saisie();
    Tant que (f(x)<f(x-pas)) faire // Attention, il faux faire un test plus malin pour détecter un minimum
        x ← x + pas
    fin tant que
    Ecrire(x);
Fin
```

Exercice 18

Écrire un algorithme demandant un entier N et un caractère et calculant puis affichant une chaîne de caractères composée de N fois ce caractère.
Dans cet exercice, il faut écrire un accumulateur de chaîne de caractères. Il faut bien insister sur le fait que la chaîne vide est l'élément neutre de l'opération concaténation.

```
Algorithme Répéter
Variables:
    N,i: entier;
    c: caractère;
    cumul: chaîne de caractères
Début
```

```
        N ← Saisie();
        c ← Saisie();
        pour i allant de 1 à N faire
                cumul ← cumul + c
        fin pour
        Ecrire(cumul);
Fin
```

Répétitives avec conditionnelles

Exercice 19

Reprendre l'exercice 16 pour afficher en plus le temps minimum au tour.
On reprend uniquement la troisième question de l'exercice 16. Il faut insister sur l'importance de choisir convenablement la valeur initiale de minimum.

```
Algorithme Formule 3 avec minimum
Variables:
        i,nbtours, temps, cumul, minimum: entiers;
Début
        nbtours ← Saisie();
        minimum ← +∞;
        Pour i allant de 1 à nbtours faire
                temps ← Saisie();
                cumul ← cumul + temps;
                si (temps < minimum et temps > 0)
                        Alors minimum ← temps;
                fin si
        fin pour
        Ecrire(nbtours);
        Ecrire(cumul);
        Ecrire(minimum);
Fin
```

Exercice 20

Sur la base de l'exercice 17, écrire un algorithme qui demande à l'utilisateur deux réels a et b (et une précision p) et affiche les minima de la fonction compris entre a et b, à p près.

```
Algorithme Maxima fonction
Variables:
        a,b,p: réel;
Début
        a ← Saisie();
        b ← Saisie();
        p ← Saisie();
```

```
    Tant que (a<b) faire
        si (f(a-pas)>f(a) et (f(a+pas)>f(a))
            Alors Ecrire('Minimum en '+a);
        a ← a + p;
    fin tant que
Fin
```

RAPPEL DE COURS

Fonctions
sin, *abs* ou *log* sont des fonctions prédéfinies (intégrées dans le langage). Il est possible de créer soi-même des fonctions ; une fois définies, ces fonctions peuvent être utilisées dans les expressions dans l'algorithme, Une fonction prend des arguments (paramètres) entre les parenthèses qui suivent son nom, et renvoie un résultat utilisé pour l'évaluation de l'expression qui l'utilise.

Définition
La définition (ou déclaration) d'une fonction explique ce que fait la fonction et comment elle le fait. Les actions n'y sont exécutées que lors d'un appel par l'algorithme. Chaque paramètre est déclaré par son nom, son type et son rôle. Les paramètres de la définition sont appelés paramètres formels.

Utilisation (appel)
Une fonction est appelée dans une expression (utilisée par exemple à droite d'une affectation, dans un affichage, derrière si, ou comme paramètre réel d'un appel de fonction) par son nom (sans le mot fonction) suivi entre parenthèse des expressions à lui fournir (paramètres réels) . Lors de l'évaluation d'une expression contenant un appel de fonction, les paramètres réels sont évalués puis transmis aux paramètres formels dans le même ordre, puis les actions de la définition de la fonction sont exécutées et le résultat retourné remplace dans l'expression à évaluer l'appel de la fonction.

Bibliothèques
Pour pouvoir utiliser dans un algorithme une fonction déjà écrite, il suffit de connaître l'entête de la fonction, c'est à dire son nom, la liste de ses paramètres formels, le type de résultat retourné et sa spécification. Une bibliothèque est une liste de fonctions dont les entêtes sont fournies et que l'on peut ainsi utiliser dans un algorithme.

Utilisation de fonctions

Exercice 21

On suppose disposer (et ce sera le cas en Javascript) des fonctions `enEntier`, `enRéel`, `enChaîne` qui convertissent respectivement une chaîne en entier, une chaîne en réel et un nombre (entier ou réel) en chaîne. Par exemple, `enEntier('123')` fournit l'entier 123 et `enChaîne(987.65)` fournit la chaîne '987.65'.

- Écrire un algorithme demandant à l'utilisateur séparément le jour (1 à 31), le mois (1 à 12) et l'année (sur 4 chiffres) d'une date puis calculant et affichant la date sous la forme jj/mm/aaaa. (utiliser la concaténation).
- Compléter l'algorithme âge1 (exercice 4) pour afficher un message « propre » comme résultat (du genre « vous avez 20 ans »).

Pas de correction pour cet exercice élémentaire.

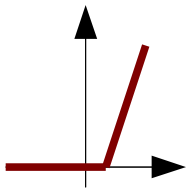
Exercice 22

On suppose disposer (et ce sera le cas en Javascript) des fonctions partie entière (notée E(...)) et valeur absolue (notée avec des traits verticaux ou abs(...))

- Écrire un algorithme demandant à l'utilisateur une longueur en cm et affichant cette longueur en pieds et pouces (remarquer qu'un pied contient 12 pouces et qu'un pouce vaut environ 2,54 cm).

```
Algorithme Conversion
Variables:
    cm,pieds,pouces: réels;
Début
    cm ← Saisie();
    pouces ← cm / 2.54;
    pieds ← pouces div 12;
    pouces ← pouces - 12 * pieds;
    Ecrire(enChaine(cm)+' centimètres = '+enChaine(pieds)+' pieds et '+enChaine(pouces)+' pouces');
Fin
```

- Écrire un algorithme modélisant la caractéristique électrique d'une diode (schéma ci-contre) : l'intensité qui traverse la diode est nulle en dessous d'une certaine valeur U_0 de la tension à ses bornes, elle suit au delà une relation linéaire $I=(U-U_0)/R$. Les valeurs de U_0 et R seront fixées dans l'algorithme, la tension sera demandée à l'utilisateur et l'intensité correspondante affichée.



```
Algorithme Diodes
Variables:
    I,U,U0,R: réels;
Début

    U0 ← 12; // 12 volts
    R ← 50; // 50 ohms
    U ← Saisie();
    I ← (U-U0)/R;
    Ecrire('Intensité = '+enChaine(I));
Fin
```

Création de fonctions

Exercice 23

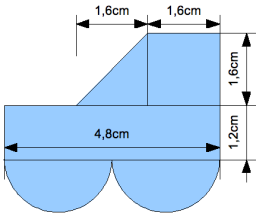
Transformer les algorithmes des exercices 5 à 9 (discussion à prévoir) en fonctions n'effectuant ni saisie, ni affichage, les données d'entrée étant passées en paramètres, le résultat renvoyé comme résultat de la fonction.

Le plus important est de donner une méthode simple de transformation de l'algorithme en fonction. En CM, on a incité les étudiants à découper l'écriture des algorithmes en 3 parties (saisies, calculs et affichage). Pour les algorithmes qui suivent ce schéma, la transformation est quasi une traduction: les saisies donnent les paramètres de la fonction, les calculs restent tels quels et les affichages (l'affichage) donne la valeur à renvoyer… Un exemple sur l'exercice 5.

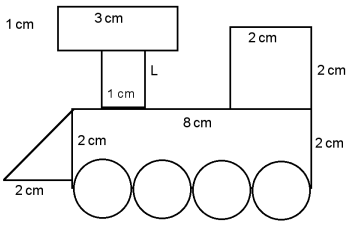
```
Algorithme Calcule age
Variables:
        age, année: entiers;
Début
        année ← Saisie();
        age ← 2010 - année;
        Ecrire(age);
Fin
```

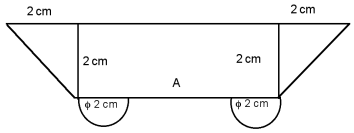
```
Fonction Calcule_age(année: entier): entier
Variables:
        age: entiers;
Début
        age ← 2010 - année;
        retourner(age);
Fin
```

Exercice 24



- Écrire une fonction donnant l'aire d'un carré dont la longueur des côtés est donnée en paramètre. En écrire une autre pour un disque à partir de son diamètre. En écrire une troisième pour un triangle dont on connaît les longueurs des 3 côtés (RAPPEL : si p est le demi-périmètre du triangle de côtés a,b, c, alors l'aire S vaut $\sqrt{(p(p-a)(p-b)(p-c))}$.). En écrire une quatrième pour un rectangle.





- Écrire ensuite un algorithme effectuant des appels judicieux de certaines de ces fonctions pour calculer puis afficher l'aire totale du dessin ci-contre. Utiliser pour les calculs intermédiaires des variables avec des noms judicieusement choisis.
- Écrire de même des algorithmes pour les figures ci-dessous

```
Fonction AireCarré(x:réel):réel
Variables:
        res:réel;
Début
        res ←x*x;
        retourner(res);
Fin
```

```
Fonction AireDisque(D:réel):réel
Variables:
        res:réel;
Début
        res ←3.14*D*D/4;
        retourner(res);
Fin
```

```
Fonction AireTriangle(a,b,c:réel):réel
Variables:
        p,res:réel;
Début
        p ←(a+b+c)/2;
        res ←sqrt(p*(p-a)*(p-b)*(p-c));
        retourner(res);
Fin
```

```
Fonction AireRectangle(l,h:réel):réel
Variables:
        res:réel;
Début
        res ←l*h;
        retourner(res);
Fin
```

```
Algorithme voiture
Variables:
        aire:réel;
```

```
Début
    aire ← AireCarre(1.6)*1.5+AireRectangle(1.2,4.6)+AireDisque(2.4);
    Ecrire(aire);
Fin
```

Exercice 25

- Écrire des fonctions effectuant les conversions de degré Celsius en Fahrenheit et réciproquement. On observe que l'eau gèle à 32°F et bout à 212°F. Bien commenter.
- Écrire aussi une fonction donnant l'équivalent en cm d'une dimension en pieds et pouces.

Utilisation de tableaux

Exercice 26

On rappelle les quelques fonctions de manipulation de tableaux suivantes:

- `Taille(<tableau>)` Retourne la taille du tableau
- `tab[<indice>]` Retourne l'élément du tableau situé à l'indice donné (le premier élément est à l'indice 0.

Dans tout cet exercice, on cherche à savoir si un élément donné figure dans un tableau.

Question 1: Recherche dans un tableau : version 1

Ecrire une fonction `estDans(<tableau>,<element>)` qui retourne la position de l'élément dans le tableau s'il y est et -1 sinon.

```
Fonction estDans(T: Tableau d'entiers, e:entier):entier
Variables:
    i,res:entier;
Début
    res ← -1;
    i ← 0;
    Tant que (i<Taille(T) et T[i] ≠ e) faire
        i ← i+1;
    fin tant que
    si (i<Taille(T))
        Alors res ← i;
    fin si
    retourner(res);
Fin
```

Combien de tours de boucle va réaliser cette fonction pour chercher une fiche qui n'est pas présente dans un tableau qui contient 70 000 000 de fiches ?

70 000 000 de tours de boucle, l'algorithme est linéaire en le nombre de fiches.

Question 2: Recherche dans un tableau : version 2

Lorsque le tableau est trié (par exemple par ordre croissant), la recherche peut-être beaucoup accélérée en considérant la stratégie dichotomique suivante: on introduit deux variables indice_début et indice_fin qui vont correspondre à la sous partie du tableau dans laquelle on va chercher l'élément. Au départ, indice_début=0 et indice_fin=Taille(<tableau>). A chaque tour de boucle, on va comparer l'élément donné à l'élément

situé exactement au milieu entre indice_début et indice_fin. S'il est inférieur, alors l'élément se trouve forcément dans la première moitié du tableau, sinon il se trouve dans la deuxième moitié du tableau. Dans les deux cas, cela implique un changement de la valeur d'un des deux indices.

1) Quel est le test d'arrêt ?

On peut s'arrêter dès qu'il n'y a plus qu'une seule case à tester et dans ce cas, soit c'est l'élément, soit ça ne l'est pas…. NB: on peut encore améliorer ce test mais cela ne change pas la complexité dans le pire des cas.

2) Quel type de boucle est le plus adapté et pourquoi ?

Une boucle Tant que car on ne connaît pas à l'avance le nombre de tours de boucle (ou plutôt, il n'est pas évident…).

3) Ecrire une fonction `estDans_rapide(<tableau>,<element>)` qui retourne la position de l'élément dans le tableau s'il y est et -1 sinon. Cette fonction sera évidemment basée sur la méthode décrite.

```
Fonction estDans_rapide(T: Tableau d'entiers, e:entier):entier
Variables:
    indice_début, indice_fin, indice_milieu,res:entier;
Début
    indice_début ← 0;
    indice_fin ← Taille(T);
    Tant que (indice_fin>indice_début) faire
        indice_milieu ← (indice_début+indice_fin) div 2;
        si (T[indice_milieu]<e)
            Alors indice_début ← indice_milieu;
            Sinon indice_fin ← indice_milieu;
        fin si
    fin tant que
    si (T[indice_début] = e)
        Alors res ← indice_début;
        Sinon res ← -1;
    fin si
    retourner(res);
Fin
```

4) Combien de tours de boucle va réaliser cette fonction pour chercher une fiche qui n'est pas présente dans un tableau qui contient 70 000 000 de fiches ?

La réponse est évidemment $\log_2(70\ 000\ 000) \simeq 26$ qu'il faut illustrer en leur faisant calculer ce nombre de tours pour 4, 8 ou 16 cases.

Réseaux Sociaux

RAPPEL DE COURS

Historique

Les réseaux sociaux sont utilisés par les sociologues pour étudier des phénomènes de groupes. Aujourd'hui le même terme désigne des environnements informatiques dans lesquels les individus s'inscrivent, indiquent certaines préférences comportementales et se déclarent en relation avec d'autres individus. Le réseau sert ensuite à mettre des individus en relation avec d'autres personnes (suivant le principe « les amis de mes amis… »). Parmi les sites les plus connus citons *facebook, myspace, twitter*. On notera que maintenant tout le monde (par exemple les partis politiques) veut son réseau social.

Vocabulaire et notations

Il s'agit de connecter des individus par une relation *A_Pour_Ami*. On notera les individus par des minuscules (a pour alain). L'ensemble de tous les étudiants est noté *E*. Pour indiquer qu'alice a bob pour ami, on notera $(a,b) \in A\_Pour\_Ami$. *A_pour_Ami* est donc un ensemble de paires *(x,y)*. Inversement on notera $(a,c) \notin A\_Pour\_Ami$ puisque colin n'est pas dans la liste des amis d'alice. Attention : la relation n'est donc pas symétrique !

Voici ci-dessous notre réseau social. Nous avons indiqué en extension les relations entre les 15 étudiants de Licence. A la ligne correspondant à denis et dont la colonne est indicée par greg on trouve un 1, ce qui indique que denis compte greg parmi ses amis (notons que ce n'est pas réciproque !).
Par exemple, les amis de colin sont $\{x \in E : (colin,x) \in A\_Pour\_Ami\} = \{a, b, c, g, i, l, n\}$.

A_Pour_Ami	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
alice	1	1	0	1	0	0	1	0	0	0	0	1	0	1	0
bob	1	1	1	1	0	0	1	0	1	1	0	1	0	1	0
colin	1	1	1	0	0	0	1	0	1	0	0	1	0	1	0
denis	1	1	0	1	0	0	1	0	0	1	1	1	0	1	0
eudes	1	1	0	1	0	0	1	0	0	1	0	1	1	1	1
françoise	1	1	0	1	1	0	1	0	0	0	0	1	0	0	0
greg	1	0	0	0	1	1	1	1	1	0	1	1	0	1	0
henri	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0
irène	1	1	0	1	0	1	0	1	1	0	1	1	0	1	0
jean	1	0	1	0	0	1	1	1	0	1	0	0	1	1	1
kamel	1	0	0	0	1	1	1	1	1	0	0	1	0	0	1
lauriane	1	0	0	1	1	0	1	0	1	1	0	1	0	0	1
marc	1	0	1	1	1	0	0	0	1	1	0	1	0	0	1
norbert	1	0	1	0	0	1	1	1	0	1	1	0	0	0	0
octave	0	1	0	0	1	0	1	0	1	1	0	0	1	1	1

La première propriété concerne l'égocentrisme… Ainsi pour dire que toutes les personnes d'un groupe (noté *G*) dont égocentriques (sont amis de soi-même) s'exprime ainsi (par une formule logique) :

$\forall x \in G, (x,x) \in A\_Pour\_Ami$. Dans ce cas, on dira que la relation *A_Pour_Ami* est **réflexive** sur *G*.

Notations associées : si *R* est une relation on notera :

- $R^{-1} = \{(x,y) : (y,x) \in R\}$, la relation inverse,
- ${}^{c}R = \{(x,y) : (x,y) \notin R\}$, la relation complémentaire.

Exercice 1

Quels sont les ensembles suivants ? Donnez une définition « en toutes lettres » et une autre définition en extension. Dans cet exercice, ça vaut vraiment la peine de parler de boucles, de boucles doubles, etc,… lorsqu'il s'agit de calculer l'ensemble en extension. ça rattache ce TD sur les réseaux sociaux aux TD déjà réalisés.

- $\{x \in E : (x,y) \in A\_Pour\_Ami \Rightarrow x=y\}$
  - *Les gens qui n'aiment personne à l'exception éventuelle d'eux mêmes (les misanthropes) = …*
- $\{x \in E : (x,x) \notin A\_Pour\_Ami\}$
  - *Les gens qui ne s'aiment pas*
- $\{x \in E : (x,colin) \notin A\_Pour\_Ami \text{ et } (bob,x) \in A\_Pour\_Ami\}$
  - *Les amis de Bob qui aiment Colin*
- $\{x \in E : |\{y:(x,y) \in A\_Pour\_Ami\}| > 7\}$
  - *Les gens qui ont strictement plus de 7 amis*
- $\{x \in E : |\{y:(x,y) \in A\_Pour\_Ami\}| > |\{y:(y,x) \in A\_Pour\_Ami\}|\}$
  - *Les gens qui ont plus d'amis que de gens qui les apprécient.*

Exercice 2

Définissez les ensembles suivants, dans chaque cas avec l'aide d'une formule :

- Les gens qui sont amis de colin et de denis
- Les gens qui sont amis de colin mais pas de denis
- Les gens qui sont amis d'un ami de denis
- Les gens qui aiment bien denis mais pas henri.

Exercice 3

Une personne sera dite *sympathique* si elle est amie de tous ceux qui sont ses amis. Comment définissez-vous cette propriété avec une formule adaptée ?

Exercice 4 : Le plus populaire

Supposons que vous cherchez à cibler les personnes les plus populaires du groupe. L'objectif par exemple peut être de leur vendre un produit en partant du principe bien connu que si xavier (très populaire) l'utilise alors il convaincra les autres. On essaie de trouver une définition de ce qu'est être populaire dans ce contexte. On notera qu'on peut être populaire parce qu'on a beaucoup d'amis, qu'on est ami de beaucoup de personnes ou une combinaison des deux.
Voici certaines définitions. Trouver pour chaque cas une raison pour laquelle cette définition ne vous va pas.

- $x : |\{y \in E : (y,x) \in A\_Pour\_Ami\}|$ est maximal
- $x : \forall y \in E\ (y,x) \in A\_Pour\_Ami$
- $x : |\{y \in E : (x,y) \in A\_Pour\_Ami\}|$ est maximal
- $x : |\{y \in E : (x,y) \in A\_Pour\_Ami\}| * |\{y \in E : (y,x) \in A\_Pour\_Ami\}|$ est maximal

Proposez éventuellement une solution alternative.

Exercice 5

Décrivez les propriétés de **symétrie** et de **transitivité** (sur un sous-ensemble d'étudiants) avec des formules logiques concernant la relation A_Pour_Ami : Est-ce que notre réseau social est symétrique ? transitif ? On notera que notre réseau est symétrique si l'amitié est symétrique sur le groupe étudié et qu'il est transitif si la maxime « les amis de mes amis sont mes amis » est vérifiée.

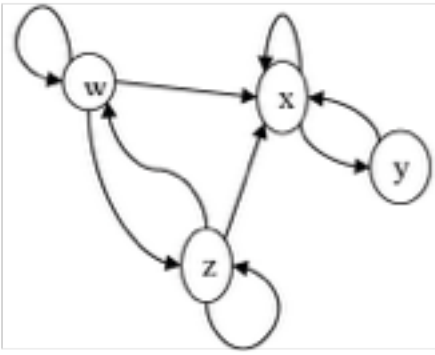
Exercice 6

On définira également, pour un sous ensemble d'étudiants *G* :
- *A_Pour_Ami* est **irréflexif** sur *G* si $\forall x \in G\ (x,x) \notin A\_Pour\_Ami$;
- *A_Pour_Ami* est **antisymétrique** sur G si $\forall x,y \in G\ (x,y) \in A\_Pour\_Ami$ et $(y,x) \in A\_Pour\_Ami \Rightarrow x=y$;
- *A_Pour_Ami* est **asymétrique** sur *G* si $\forall x,y \in G\ (x,y) \in A\_Pour\_Ami \Rightarrow (y,x) \notin A\_Pour\_Ami$.

Trouvez des groupes de 3 étudiants tels que : A_Pour_Ami, restreint au groupe est :
- irréflexif et transitif (on appelle ça une relation d'ordre strict) ;
- réflexif, antisymétrique et transitif (on appelle ça une relation d'ordre large) ;
- réflexif, symétrique et transitif (on appelle ça une relation d'équivalence) ;
- ni réflexif, ni irréflexif, ni symétrique, ni asymétrique, ni antisymétrique, ni transitif.

Exercice 7 : Graphe

Avec la matrice ci-contre, on associe un dessin dans le plan appelé graphe.



A_pour_Ami	w	x	y	z
william	1	1	0	1
xavier	0	1	1	0
yann	0	1	0	0
zébulon	1	1	0	1

Dessinez le graphe correspondant à la partie du réseau limité à d, e, f, g et h.

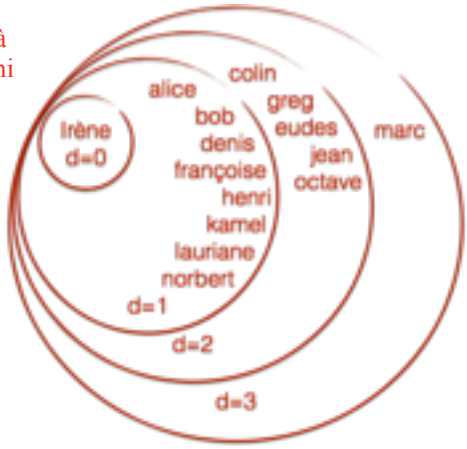
Exercice 8

En utilisant les termes (de théorie des graphes) boucle, arc, sommet, chemin, circuit (et qu'on ne détaillera pas ici), définissez les termes de réflexivité, symétrie, antisymétrie, transitivité.

Exercice 9

On cherche à établir la distance moyenne entre 2 personnes comme étant le nombre minimum d'amis par lesquels il faut passer pour aller de l'un à l'autre. (1) Calculer la distance entre irène et marc. (2) Quelles propriétés concernant cette mesure de distance pouvez-vous énoncer ? (3) Comment pourriez-vous calculer cette distance ?

(1) Le but de cette question est de laisser travailler les étudiants. Ils parviendront ensuite à trouver que la distance est de 3 (par exemple Irène a pour ami Françoise qui a pour ami Eudes qui a lui-même Marc pour ami mais il y a plein d'autres chemins possibles).
(2) C'est une distance et donc elle vérifie l'inégalité triangulaire….
(3) On peut parler ici d'algorithme de Dijkstra mais il ne s'agit pas d'en donner l'algorithme. Il y a tout de même moyen de discuter autour de cet algorithme et de calculer les distances d'Irène à tous les autres en procédant par cercles successifs.



Exercice RS.10

Nous allons à nouveau analyser le réseau social (alice, bob et leurs amis,… norbert, octave). Il s'agit donc d'un groupe de 15 individus, reliés par la relation a_pour_ami.
Un premier tableau, appelé **Noms**, contient les différentes personnes (alice est le 0, octave le 14).

On suppose que Noms a été déclaré par
Noms : Tableau de 14 chaînes de caractères
et qu'il y a eu une instruction du type
Noms ← ['Alice', 'Bob', …, 'Norbert'];

- Ecrire la fonction prénom qui étant donné un numéro, retourne le prénom de la personne

La fonction Prenom est donc évidente, il s'agit d'accéder à la bonne case du tableau Noms.

```
Fonction Prenom(nb : entier) : chaîne de caractères
Début
        retourner Noms[nb];
Fin
```

- Ecrire la fonction numéro qui, étant donné un nom, retourne le numéro correspondant.

Il s'agit de parcourir le tableau Noms, on renverra -1 lorsqu'il n'y a pas de prénom correspondant.
Pour cette question, on peut fournir plusieurs solutions car il s'agit d'une recherche dans un tableau trié. Cependant, dans l'évaluation, on se limitera à la connaissance d'une solution non dichotomique.

```
Fonction Numero_v1(ch : chaîne de caractères): entier
// On ne fait pas l'hypothèse que le tableau est trié et on le parcours entièrement quoi qu'il arrive
Variables:
        i: entier;
        res: chaîne de caractères
Début
        res ← -1;
        pour i allant de 0 à Taille(Noms)-1 faire
                si (ch = Noms[i])
                        alors res ← i;
                fin si
        fin pour
        retourner res;
Fin

Fonction Numero_v2(ch : chaîne de caractères): entier
// On ne fait pas l'hypothèse que le tableau est trié et s'arrête dès qu'on a trouvé l'élément
Variables:
        i: entier;
        res: chaîne de caractères
Début
        res ← -1;
        i ← 0;
        tant que (ch ≠ Noms[i] et i<Taille(Noms)) faire
                i ← i +1;
        fin tant que
        si (i<Taille(Noms))
                alors res ← i;
        fin si
        retourner res;
Fin
```

```
Fonction Numero_v3(ch : chaîne de caractères): entier
// On fait l'hypothèse que le tableau est trié et s'arrête dès qu'on a trouvé l'élément
Variables:
        i: entier;
        res: chaîne de caractères
Début
        res ← -1;
        i ← 0;
        tant que (ch < Noms[i] et i<Taille(Noms)) faire
                i ← i +1;
        fin tant que
        si (i<Taille(Noms) et ch = Noms[i])
                alors res ← i;
        fin si
        retourner res;
Fin

Fonction Numero_v4(ch : chaîne de caractères): entier
// On fait l'hypothèse que le tableau est trié et on fait une recherche dichotomique
Variables:
        debut, milieu, fin: entier;
        res: chaîne de caractères
Début
        res ← -1;
        tant que (abs(fin-debut)>1) faire
                milieu ← (debut+fin) div 2;
                si (ch < Noms[milieu])
                        alors fin ← milieu;
                        sinon debut ← milieu;
                fin si
        fin tant que
        si (ch = Noms[debut])
                alors res ← debut;
                sinon  si (ch = Noms[fin])
                                alors res ← fin;
                        fin si
        fin si
        retourner res;
Fin
```

On suppose maintenant que les données du tableau ont été stockées dans un tableau à deux dimensions T, en utilisant une numérotation des personnes de 0 à 14. Ainsi T[0,6] vaut 1 puisque alice aime bien greg.

- Quelle est la déclaration de ce tableau ?

```
T : Tableau de taille 14x14 de booléens
```

- Ecrire un algorithme qui affiche tous les amis de lauriane (les gens que lauriane aime bien)

```
Algorithme amis de Lauriane
Variables:
        num, i: entier;
Début
        num ← Numero('Lauriane');
        pour i allant de 0 à 14 faire
                si (T[num,i])
                        alors Ecrire(Prenom(i));
                fin si
        fin pour
Fin
```

• En faire une fonction qui affiche les prénoms de tous les amis d'une personne dont on aura fourni le prénom.

```
Fonction Amis_de(ch:chaîne): rien
Variables:
        num, i: entier;
Début
        num ← Numero(ch);
        pour i allant de 0 à 14 faire
                si (T[num,i])
                        alors Ecrire(Prenom(i));
                fin si
        fin pour
Fin
```

• Construire un algorithme qui compte le nombre d'amis de lauriane.

```
Algorithme Nombre d'amis de Lauriane
Variables:
        num, i, cpt: entier;
Début
        num ← Numero('Lauriane');
        cpt ← 0;
        pour i allant de 0 à 14 faire
                si (T[num,i])
                        alors cpt ← cpt +1;
                fin si
        fin pour
        Ecrire(cpt);
Fin
```

• Construire un algorithme qui trouve la personne qui a le plus d'amis.

```
Fonction Nombre_amis(ch:chaîne de caractères) : entier
Variables:
        num, i, cpt: entier;
Début
        num ← Numero(ch);
        cpt ← 0;
        pour i allant de 0 à 14 faire
                si (T[num,i])
                        alors cpt ← cpt +1;
                fin si
        fin pour
        Retourner cpt;
Fin

Algorithme Nombre max d'amis
Variables:
        i, imax, max, nb: entier;
Début
        imax ← 0;
        max ← Nombre_amis(Prenom(0));
        pour i allant de 1 à 14 faire
                nb ← Nombre_amis(Prenom(i));
                si (nb > max)
                        alors  max ← nb;
                                imax ← i;
                fin si
        fin pour
        Ecrire(Prenom(imax));
Fin
```

Algorithmique Distribuée

Exercice 1: Partage de données

Deux algorithmes indépendants souhaitent effectuer des calculs sur des variables communes, ici x et y contenant des entiers. Voici les deux algorithmes concurrents (i.e. qui s'exécute en même temps) :

Algorithme 1

```
1.  x ← x + 1 ;
2.  y ← y + x ;
```

Algorithme 2

```
3.  x ← x + 2 ;
4.  y ← y - x ;
```

Supposons que x et y ont été initialisées à 0.

- Sachant que les deux algorithmes ne sont pas synchronisés entre eux, combien d'exécutions sont possibles ?
- Quels sont les valeurs finales possibles pour x et y ?

Pour éviter que les valeurs des variables changent pendant qu'un algorithme s'exécute, nous faisons intervenir une entité tierce qui empêchera l'accès à une variable tant qu'un autre algorithme est déjà en cours d'accès. Ce mécanisme s'appelle l'exclusion mutuelle.

Ainsi, avant d'utiliser une variable commune (couramment appelée variable partagée), les algorithmes vont demander la permission : c'est la fonction `Obtenir(variable)`.

De même, quand ils ont fini de l'utiliser, ils libèrent l'accès à la variable partagée : c'est la fonction `Libérer(variable)`. Par conséquent, si un algorithme demande a obtenir une variable qui est déjà utilisée par un autre algorithme, il sera mis en attente jusqu'à ce que l'autre algorithme libère la variable.

On considère maintenant les deux algorithmes modifiés suivants. Chaque algorithme s'exécutera seul et intégralement,

Algorithme 1

```
1.  Obtenir(x) ;
2.  Obtenir(y) ;
3.  x ← x + 1 ;
4.  y ← y + x ;
5.  Libérer(y) ;
6.  Libérer(x) ;
```

Algorithme 2

```
7.  Obtenir(y) ;
8.  Obtenir(x) ;
9.  x ← x + 2 ;
10. y ← y - x ;
11. Libérer(x) ;
12. Libérer(y) ;
```

sans que les valeurs de `x` et `y` ne soient changées par l'autre :

- Que se passe-t-il si on a l'ordre d'exécution suivant : `1 → 7 → 2 → 8 →` ... ?
- Comment peut-on modifier les algorithmes pour éviter ce problème ?

Exercice 2 : Le diner des philosophes

Régulièrement, cinq philosophes amis se retrouvent autour d'un bon repas pour réfléchir ensemble sur le sens de la vie. Le couvert est toujours disposé comme sur la figure ci-après. Une fois installés dans leur fauteuil, nos philosophes ne peuvent être que dans 3 états : pensant, affamé et mangeant.

-Un philosophe ne peut manger et penser en même temps ;

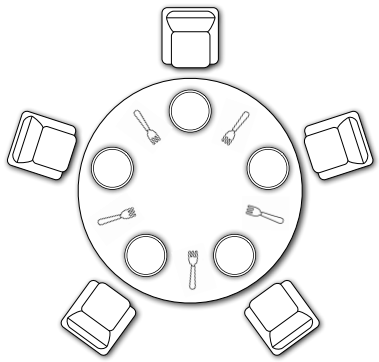
-Un philosophe pensant peut devenir affamé à n'importe quel moment ;

-Un philosophe affamé cherche à se mettre à manger ;

-Pour manger, un philosophe doit avoir le contrôle des deux fourchettes autour de lui ;

-Après avoir mangé, un philosophe repose les fourchettes et se remet à penser.

Considérons tous les philosophes comme similaires. L'algorithme ci-dessous coordonne l'ordre des actions d'un philosophe.



Algorithme Philosophe
1. **Tant que** `vrai` **faire**
2. `Penser();`
3. `Obtenir(fouchetteGauche);`
4. `Obtenir(fouchetteDroite);`
5. `Manger();`
6. `Libérer(fouchetteDroite);`
7. `Libérer(fouchetteGauche);`
8. **Fait**

- Interblocage Trouver un historique d'exécution qui abandonne tous les philosophes en état affamé.

On considère qu'on dispose des deux fonctions suivantes, qui prennent en paramètre un nom de fourchette : `estLibre` retourne un booléen indiquant si cette fourchette n'est pas en cours d'utilisation par un autre philosophe et `enPossession` retourne un booléen indiquant si cette fourchette est en cours d'utilisation par le philosophe exécutant l'algorithme.

- Comment modifier l'algorithme pour éviter les interblocage ?
- Famine À partir de l'algorithme modifié, trouver un historique d'exécution qui affame éternellement un philosophe.

Exercice 3: En direct de Waterloo

Plusieurs maréchaux de la vieille garde sont éparpillés dans la campagne belge. Ils doivent décider s'ils vont attaquer les anglais ou s'il vont battre en retraite. Ils communiquent par l'intermédiaire de sergents à cheval qui prennent un temps indéterminé pour effectuer leur trajet. Cependant, il est possible que les anglais fassent prisonnier un maréchal, mais il n'est pas possible de savoir quand, ni lequel. Tous les sergents envoyés vers un maréchal prisonnier deviennent prisonniers aussi. Chaque maréchal a une idée initiale de ce qu'il voudrait faire.

Il faut que les propriétés suivantes soient respectées :

Terminaison Tous les maréchaux qui ne sont pas prisonniers, prennent une décision ;

Consensus Tous les maréchaux qui prennent une décision prennent la même décision ;

Validité Si tous les maréchaux sont du même avis initial, celui-là doit correspondre à la décision finale.

- Dans un premier temps, nous considérons que les sergents sont furtifs et les maréchaux insaisissables. Comment faire pour que tous les maréchaux connaissent l'opinion de chacun ? En déduire une solution du problème dans ce cas.
- On lève l'hypothèse de sergents furtifs et maréchaux insaisissables, mais on connait le temps maximum mis par un sergent pour délivrer un message. Trouver une solution au problème et gagnez la bataille de Waterloo !

Applications en Bioinformatique

Traitement de chaînes

Exercice 1 : Manipulation génétique

La transcription permet de passer d'une séquence d'ADN (chaîne de caractères composée de A, T, C et G) à une séquence d'ARN. Ce processus biologique consiste à "changer" les T en U dans une chaîne d'ADN.

- Proposer un algorithme qui demande une séquence d'ADN à l'utilisateur et qui affiche la séquence d'ARN correspondante. Vous disposez pour cela de la fonction `CaractereEn(<chaine>,<pos>)` qui retourne le caractère d'une chaîne de caractères `<chaine>` à une position `<pos>`.

- Envisager une autre version qui inclue une vérification de la séquence d'ADN, i.e. si un caractère n'est pas A, T, C, ou G, ce n'est pas une séquence d'ADN et la transcription s'arrête.

Exercice 2 : Complément

```
ATCGTA
TAGCAT
```

La séquence d'ADN est constituée d'une double hélice constituée de deux brins complémentaires : A et T sont complémentaires, ainsi que G et C.
Proposez l'algorithme d'une fonction qui donne le brin (i.e. la séquence) complémentaire d'une chaîne ADN passée en paramètre. Exemple ci-contre :

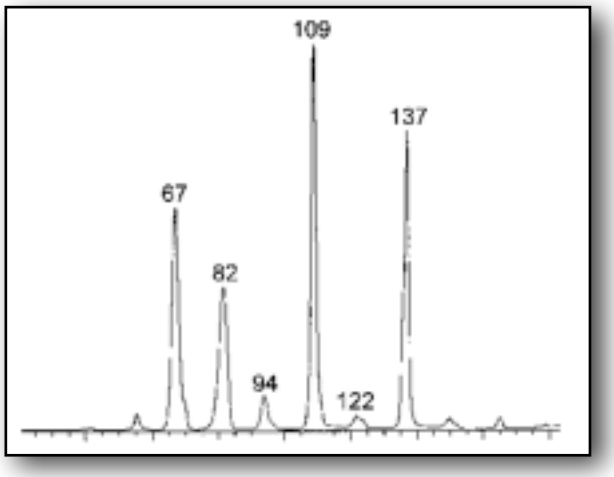
Exercice 3 : Les experts Las Vegas

Un homicide a eu lieu dans le désert à 55 km de Las Vegas. Sur les lieux du crime, la police scientifique de Las Vegas a retrouvé une pierre avec des traces de sueur. A partir de cet échantillon, il est ainsi possible d'isoler un fragment d'ADN.

Grissom vous charge de comparer ce fragment d'ADN avec celui de la victime. On supposera que les chaînes d'ADN sont des chaînes de caractères. Ecrire l'algorithme qui indiquera si l'ADN trouvé sur le site est éventuellement une sous-chaîne entière de l'ADN de la victime (i.e. un mot inclus entièrement dans un autre mot). On arrêtera la recherche dès lors que l'on aura identifié la sous-chaîne dans l'ADN de la victime.

Traitement de tables

Exercice 4 : Les experts Miami



L'analyse de substances chimiques constitue une étape primordiale de la recherche scientifique industrielle et fondamentale. La spectrométrie de masse indique des intensités différentes pour une gamme de longueur d'ondes donnée. Ainsi, chaque substance est caractérisée par des pics d'intensité donnés à des positions (longueur d'onde) données. On cherche à isoler automatiquement les pics issus d'une analyse par spectrométrie de masse.

Cette information est stockée dans un tableau nommé `Intensite_tab` qui donne les valeurs d'intensités pour un poids (indice) donné. On suppose que le intensités de tous les poids entre 0 et une valeurs maximale sont données (i.e., dans l'exemple, on connait les intensités pour les poids 67, 82, 94,... etc mais aussi tous les poids intermédiaires entre 0 et 180).

- Proposer un algorithme qui donne la valeur maximale des intensités ?
- Proposez un algorithme qui recherche les pics d'intensités ? On commencera par afficher les indices de ces pics d'intensité, puis on stockera dans un tableau les intensités de ces pics.
- Afin de faire un filtre parmi les pics d'intensités, améliorez le précédant algorithme en ajoutant la possibilité à l'utilisateur de prendre en compte un seuil d'intensité (`seuil`) au-dessous duquel les pics d'intensité seront négligés.

Simulation de systèmes dynamiques

Exercice 5 : Parc naturel du Mercantour

Une réserve naturelle tente de réintroduire des loups dans une zone qui se trouve totalement dépeuplée ; on réintroduit donc des loups issus d'une réserve italienne voisine. Ces loups se reproduisent avec un taux annuel de renouvellement de 15% ce chiffre tenant compte des naissances et des morts naturelles, mais pas des bergers et braconniers qui abattent environ 25 bêtes par an. Écrire un algorithme qui prend le nombre de loups introduits puis calcule et affiche au bout de combien d'années la réserve se retrouve à nouveau dépeuplée.

Exercice 6 : Informatique pour l'agro-alimentaire

La production de yaourt consiste à introduire des bactéries (Streptococcus thermophilus & Lactobacillus bulgaricus) dans un volume de lait. En conditions favorables (températures de 43 degrés et présence de sucres), les population bactériennes connaissent une croissance dite exponentielle si le milieu n'est pas limitant (i.e. présence de sucres suffisante). Le taux de croissance est alors constant. On considère un taux de croissance horaire μ égal à 2.5 pour les populations qui nous intéressent (toutes les heures la population est augmentée de 2,5% de sa valeur).

- Ecrire un algorithme qui calcule au bout de combien de temps (heures) la population aura doublé, en supposant que la mortalité est nulle.

• Une fois stocké au froid, la croissance bactérienne s'arrête. Alors, d'un point de vue pratique, chaque heure, 10% des bactéries meurent. Ecrire un algorithme qui calcule au bout de combien de temps (heures) la population bactérienne totale comporte au moins 20% de bactéries mortes pour une quantité de bactéries initiale donnée.

•**(Option)** Un produit laitier est périmé quand la population bactérienne est constituée de 50% d'individus morts. Cette estimation est utilisés pour estimer une date de péremption, telle qu'elle peut être indiquée sur l'emballage de vos yaourts. L'incubation des bactéries (croissance en milieu favorable) dure 48 heures, au bout desquelles les bactéries sont confinées dans un milieu clos et réfrigéré. Proposez un algorithme qui indique au bout de combien d'heures (incubation comprise) le yaourt sera périmé. Par mesure de commodité, le temps en heures pourra être converti en jours.

Exercice 7 - Loup suite

Dans la même réserve naturelle, on cherche maintenant à étudier l'impact réel d'une population de loups (prédateurs) sur une population de proies naturelles (les moutons). Pour ce faire, nous considérons la ressource en substrat pour les moutons comme inépuisable. On associe alors un modèle prédateur-proie de Lotka-Voltera (1925). Ce modèle classique tient compte de l'abondance de chaque espèce. C'est une simplification grossière de la réalité biologique mais qui permet cependant de faire des estimations. On considère alors les taux de variations des populations (respectivement v_x et v_y) comme suivant:

∂ le taux de croissance naturelle des moutons sans prédation (0,04),

ß le taux de mortalité des moutons en présence de prédateurs (0,0005),

μ l'efficacité d'assimilation de la biomasse de proie en prédateur (0,1),

$$\begin{aligned} v_X &= \delta - \beta Y \\ v_Y &= \mu\beta X - \gamma \end{aligned}$$

γ le taux de mortalité naturelle des prédateurs (0,2), avec X étant les moutons (proies) et Y les prédateurs (loups). Ainsi à chaque instant (t), la nouvelle population (t+1) peut être estimée comme ci contre :

$$\begin{aligned} X_{t+1} &= X_t + v_x X_t \\ Y_{t+1} &= Y_t + v_y Y_t \end{aligned}$$

Proposez un algorithme qui permet la saisie de la population initiale de proies, de prédateurs et du temps de simulation ; et qui stocke et affiche la variation de la population de proies au cours du temps. Vous proposerez 2 versions de stockage pour ces informations :

1. dans une chaîne de caractères
2. dans un vecteur.

Exercice 8 - Simulation discrète

A active B
A active C
B active D
C active D
D active A

On propose de simuler la dynamique d'un réseau de régulation de gènes. On suppose que les gènes suivent une dynamique discrète. A chaque pas de temps les gènes prennent des valeurs booléennes (0 ou 1) en fonction de l'état d'activité des autres gènes en présence. Proposez un programme javascript qui permet de reproduire la dynamique d'un petit réseau composé de 4 gènes et des 5 règles tels que :

Dès qu'un gène active un autre gène, il est désactivé, mais il peut être réactivé dans la même itération par un autre gène. Dans un premier temps, les règles sont à appliquer dans l'ordre en utilisant les nouvelles activations/désactivations induites par les règles précédentes.

On cherchera à produire une sortie d'algorithme qui tienne compte du nombre d'itérations, correspondant à l'exemple ci-dessous.

0|1000	0|1111	0|1010
0|1000	0|1000	0|1000
1|1000	1|1000	1|1000
2|1000	2|1000	2|1000
3|1000	3|1000	3|1000

Comment intégrer le fait qu'un gène actif puisse inhiber l'activité d'un autre gène ?

Complétez votre algorithme pour y intégrer, avec un critère aléatoire, la compétition entre les gènes B et C pour la ressource gène A.

Traitement Automatique des Langues

Exercice 1 :Coder / décoder

L'introduction à cet exercice provient de http ://www.bibmath.net/crypto/substi/cesar.php3.

Le code de César est la méthode de cryptographie la plus ancienne communément admise par l'histoire. Il consiste en une substitution mono-alphabétique, où la substitution est définie par un décalage de lettres. Par exemple, si on remplace A par D, on remplace B par E, C par F, D par G, etc... Donnons un exemple sur ce décalage de 3 lettres :

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
DEFGHIJKLMNOPQRSTUVWXYZABC
```

Le texte que nous souhaitons coder est le suivant :

```
JE LEVAI LES YEUX VERS LE SOLEIL IL ETAIT BAS ; DANS MOINS D'UNE HEURE IL ARRIVERAIT
JUSTE AU-DESSUS DES BRANCHES SUPERIEURES DU VIEUX CHENE.
```

Le texte codé est alors :

```
MH OHYDL OHV BHXA YHUV OH VROHLO LO HWDLW EDV ; GDQV PRLQV G'XQH KHXUH LO
DUULYHUDLW MXVWH DX-GHVVXV GHV EUDQFKHV VXSHULHXUHV GX YLHXA FKHQH.
```

Il n'y a que 26 façons différentes de crypter un message avec le code de César. Cela en fait donc un code très peu sûr, puisqu'il est très facile de tester de façon exhaustive toutes les possibilités. Pourtant, en raison de sa grande simplicité, le code de César fut encore employé par les officiers sudistes pendant la guerre de Sécession, et même par l'armée russe en 1915.

• Écrire une fonction `encode` qui prend en paramètres une chaîne représentant le message à encoder, un entier ∈ [0, 25] représentant le décalage, et qui renvoie une chaîne représentant le message codé. On suppose que tout est écrit avec des lettres majuscules non accentuées, et on laisse inchangés tous les autres caractères.

• Écrire la fonction réciproque `décode` qui prend en paramètres une chaîne représentant le message à décoder, un entier ∈ [0, 25] représentant le décalage qui a servi à encoder, et qui renvoie une chaîne représentant le message en clair.

• Écrire un algorithme qui demande un message codé selon le code de César et qui affiche toutes les tentatives de décodage jusqu'à ce que l'utilisateur lui dise que c'est la bonne, parce qu'il a reconnu un message en clair. Dans le cas où l'utilisateur ne reconnaît jamais de message en clair, l'algorithme doit s'arrêter dès que les 26 décalages possibles ont été proposés.

Exercice 2 : Fréquence des lettres

Écrire une fonction qui parcourt un texte donné en paramètre, et qui renvoie un tableau de fréquences des lettres du texte. Ce tableau aura 26 éléments, le premier élément contiendra le nombre de 'A', le deuxième le nombre de 'B', . . ., et le dernier le nombre de 'Z'.

Remarque : Dans un premier temps on ne tiendra compte que des lettres non accentuées, puis on proposera des solutions pour traiter aussi les lettres accentuées.

Exemple : Pour le texte *«Toute conception revient à utiliser de manière originale des éléments préexistants»*[1], le tableau `frequences` obtenu est présenté ci-dessous à gauche.

Intérêt : Si on traite des textes écrits dans des langues différentes, et qu'on trie les tableaux de fréquences par nombre décroissant d'occurrences (comme ci-dessous à droite), on se rend compte que chaque langue a sa propre distribution de fréquences.

1	(a)	4
2	(b)	0
3	(c)	2
4	(d)	2
5	(e)	15
6	(f)	0
7	(g)	0
8	(h)	0
9	(i)	8
10	(j)	0
11	(k)	0
12	(l)	3
13	(m)	2
14	(n)	7
15	(o)	4
16	(p)	2
17	(q)	0
18	(r)	5
19	(s)	3
20	(t)	8
21	(u)	2
22	(v)	1
23	(w)	0
24	(x)	1
25	(y)	1
26	(z)	0

fréquences

(e)	15
(i)	8
(t)	8
(n)	7
(r)	5
(a)	4
(o)	4
(l)	3
(s)	3
(c)	2
(d)	2
(m)	2
(p)	2
(u)	2
(v)	1
(x)	1
(y)	1
(b)	0
(f)	0
(g)	0
(h)	0
(j)	0
(k)	0
(q)	0
(w)	0
(z)	0

fréq. ↘

Exercice 3 : Génération de texte

On dispose d'un tableau, `animaux`, de chaînes de caractères dans lequel sont rangés les noms de `n` animaux précédés de l'article défini convenable.

Écrire une fonction qui prend ce tableau `animaux` et qui renvoie un tableau de chaînes dans lequel ont été générées toutes les questions possibles sur le modèle : **« Est-ce que le chat mange la souris ? »**. Il faudra éviter les questions du type **« Est-ce que le chat mange le chat ? »**.

[1] 1Les technologies de l'intelligence, Pierre Lévy

Extension : Modifier cette fonction (et éventuellement ses paramètres) pour tenir compte du genre en générant des phrases comme **« le chat mange-t-il la souris ? »** et **« la souris mange-t-elle le chat ? »**

Exercice 4 : Compter les mots

```
Nous sommes le 11 octobre 2008. Avant-hier Jean-Claude a épluché dix-huit
pommes de terre. En effet nous étions six convives. Cet homme-là est un
spécialiste de l'épluchage. Il se considère lui-même comme le plus rapide
du domaine. Aujourd'hui quatre pommes de terre sont en train de pourrir
car, voyez-vous, quatorze d'entre elles ont suffi à nous rassasier!
```

• Compter à la main le nombre de mots du texte précédent en notant le type de questions qui se posent.

• Proposer une définition du **mot** qui permette d'automatiser la segmentation d'un texte en mots.

• On considère que le texte est rangé dans une *variable* de type *chaîne de caractères*. On appelle *caractère alphanumérique* un caractère qui est une **lettre** (majuscule, minuscule, accentuée ou non) ou un **chiffre**. On appelle séparateur tout caractère qui n'est pas alphanumérique. Écrire un algorithme qui compte les **mots** en prenant comme définition du **mot** toute chaîne de **caractères alphanumériques** suivie par un **séparateur**. Pour simplifier on considère que le dernier caractère du texte est un séparateur. On utilisera une fonction booléenne `est_alphanum(car)` qui renvoie *vrai* si et seulement si le caractère passé en paramètre est une lettre ou un chiffre.

• Écrire la fonction booléenne `est_alphanum(car)` pour l'*anglais* puis pour le *français*.

Exercice 5 : Le mot le plus long

Écrire un algorithme qui prend un texte rangé dans une variable de type chaîne de caractères et qui calcule la longueur moyenne des mots ainsi que le mot le plus long et sa longueur. On reprendra les hypothèses de l'exercice *Compter les mots*.

Exercice 6 : Indexer

On se propose de «calculer» le vocabulaire d'un texte. L'objectif est de prendre un texte et de construire deux tableaux. Le premier tableau `lexique` représente une liste, sans répétition, de tous les mots[2] du texte. Le deuxième tableau `occur` est un tableau d'effectifs tel que $\forall i$, `occur[i]` représente le nombre d'occurrences du mot `lexique[i]` dans le texte. Par exemple, avec le texte «*Le valet de chambre accuse la cuisinière, qui accuse la lingère, qui accuse les deux autres*». (Maupassant, Le Horla), on doit obtenir les tableaux ci-contre :

Écrire un algorithme qui demande un texte (variable chaîne de caractères), et construit ces deux tableaux en utilisant la fonction `position_tab` donnée dans les annexes. On reprendra les hypothèses de l'exercice Compter les mots.

Intérêt : Si on trie le tableau `lexique` par nombre décroissant d'occurrences, on visualise les mots les plus fréquents, qui sont en général les « mots grammaticaux »[3]. Les mots qui viennent juste après dans le classe-

[2] Pour la définition du mot, se reporter à l'exercice précédent

[3] On nomme ainsi les déterminants, prépositions, conjonctions..., par opposition aux mots «à contenu»

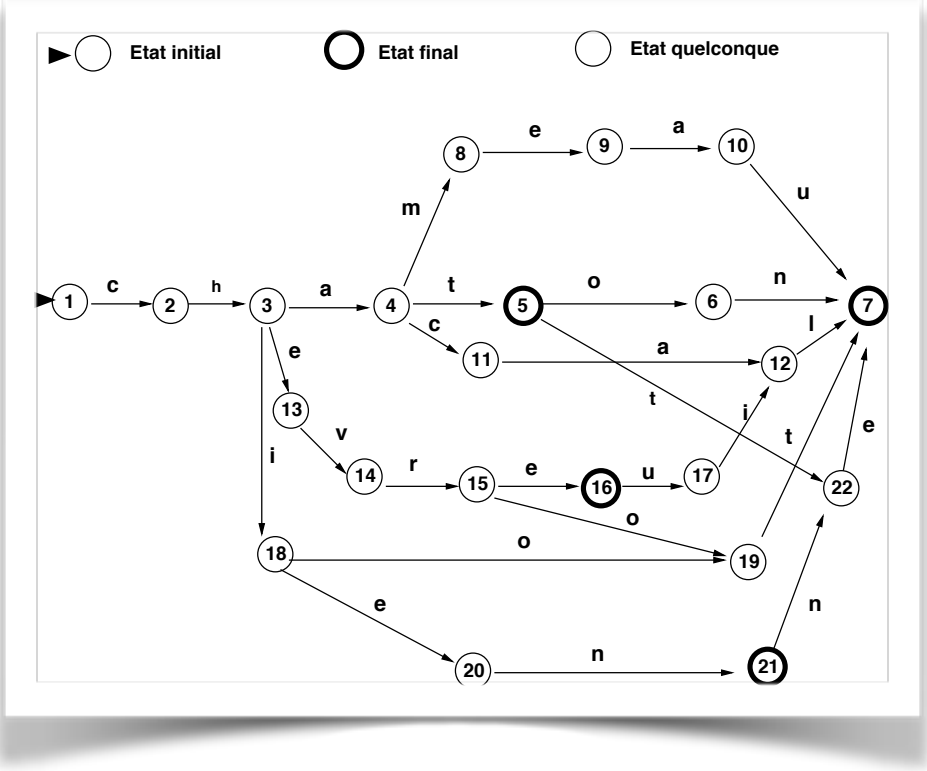
ment sont très intéressants pour l'analyse automatique du texte. Ils permettent en général de catégoriser celui-ci sur le plan du domaine, et du genre. L'échantillon de l'exemple est bien sûr trop court pour illustrer ce propos.

Exercice 7 : Automate

Un automate à nombre fini d'états est une machine abstraite utilisée pour l'étude des langages. Un automate est composé d'*états* (dont un *état initial* et un ou des *états finaux*) et de *transitions*. Son comportement (qui peut être programmé) est dirigé par un mot fourni en entrée. On se place sur l'état initial[4] et on examine la première lettre du mot. Si, de cet état, part une transition portant cette lettre, on change d'état courant en suivant la transition. On recommence avec la seconde lettre et ainsi de suite. À partir d'un mot donné, trois situations peuvent apparaître :

- La transition commandée par la dernière lettre du mot conduit sur un état final : le mot est reconnu et appartient au langage de l'automate (exemple "chat").
- La transition commandée par la dernière lettre du mot conduit sur un état non final : le mot n'est pas reconnu et n'appartient pas au langage de l'automate (exemple "chac" car l'état 11 n'est pas final).
- À partir d'un certain état il n'y a pas de transition pour la lettre examinée : le mot n'est pas reconnu et n'appartient pas au langage de l'automate (exemple "chenille" bloque sur l'état 13).

L'ensemble des mots reconnus par un automate s'appelle le *langage de l'automate*.



- Donner le chemin de l'automate qui permet de reconnaître le mot "chameau".
- Le mot "chamelle" fait-il partie du langage de l'automate ? Expliquer pourquoi.
- Le mot "chato" fait-il partie du langage de l'automate ? Expliquer pourquoi.
- Donner le langage de l'automate, en le rangeant par ordre alphabétique croissant.
- On cherche à comparer la place occupée par un automate et la place occupée par la liste des mots qu'il renferme, mémorisée de manière classique dans un tableau de chaînes. On utilise l'approximation suivante :
  - Pour évaluer la place occupée par la liste, on compte une unité mémoire par lettre.
  - Pour évaluer celle prise par l'automate, on compte une unité mémoire par état, et une unité mémoire par transition.

[4] On se limite aux automates déterministes

Comparer la place occupée par la liste de mots et celle occupée par l'automate, selon cette approximation.

•On cherche à comparer le temps mis à trouver un mot dans la liste classique et dans l'automate. On utilise l'approximation suivante :

-Pour évaluer le temps mis à trouver un mot dans la liste, on considère que les mots sont classés par ordre alphabétique croissant et on compte une unité de temps par lettre à parcourir pour trouver le mot (on abandonne le parcourt d'un mot dès qu'on est sûr que ce n'est pas lui).

-Pour évaluer le temps mis à trouver un mot dans l'automate, on compte une unité de temps par transition à examiner à partir de chaque état visité (si trois flèches partent d'un état, il faut compter 3 unités de temps). Comparer le temps mis à trouver "chacal", "chiot", et à ne pas trouver "chenille'.

•Dessiner un automate dont le langage est constitué par les six formes conjuguées du verbe "jouer" au présent de l'indicatif.

Exercice 8 :Chercher une chaîne correspondant à une expression rationnelle

•Trouver des mots de la langue française qui sont reconnus par les expressions rationnelles suivantes :

1. `a.+tion` 2. `ch.{2-4}` 3. `ils ?|elles ?`

•Définir les langages (ensembles de chaînes qui ne constituent pas nécessairement les mots d'une langue) définis par les expressions rationnelles suivantes :

1. `(a|b)*` 2. `a*|b*` 3. `a*b*` 4. `ab*a`

Exercice 9 : Écrire des expressions rationnelles

Écrire les expressions rationnelles qui décrivent des chaînes de caractères correspondant aux patrons suivants :

• mots qui commencent par 'M', finissent pas 'E' et dont la longueur est comprise entre 2 et 5

• les mots qui contiennent 2 't' consécutifs

• les nombres entiers écrits en binaire

• les chaînes représentant un numéro de téléphone français dans le format national ou dans le format international (commencent par +33)

• les chaînes représentant une date au format (JJ/MM/AAAA). Il ne s'agit pas de contrôler la validité de la date, mais seulement son format.

• les chaînes composées de tranches de trois chiffres séparées par un point (exemple : 123.563.886.555) 7. les nombres décimaux (les entiers en font partie)

Annexes

Caractères : `CaractereVersAscii(car : caractère) : entier`
renvoie le code ASCII du caractère `car` ◅ exemple : `Caractere_vers_Ascii('A')` renvoie 65, `Caractere_vers_Ascii('B')` renvoie 66 (deux lettres consécutives dans l'alphabet ont comme codes deux entiers consécutifs)
`Ascii_vers_Caractere(code : entier) : caractère` renvoie le caractère codé par `code` ◅ exemple `char(65)` renvoie ' A'
`majuscule(car : caractère) : caractère` renvoie la majuscule si `car` est une lettre (accentuée ou non), ou le caractère inchangé sinon
Chaîne de caractères : Comme en Javascript et en C on considère que le premier caractère est à la position 0.
`ch[i]` est le caractère de `ch` qui se trouve à la position i.
`Longueur(ch : chaîne) : entier` renvoie le nombre de caractères de `ch` ◅ exemple : `longueur("Au revoir !")` renvoie 11
`PositionDans(ch : chaîne; car : caractère) : entier` renvoie la position de la première occurrence de `car` dans `ch`, ou -1 si `car` est absent de `ch` ◅ exemple : `position_ch('abcde','c')` renvoie 2
`SousChaine(ch : chaîne; début, longueur : entier) : chaîne` renvoie la chaîne de longueur `longueur`, extraite de `ch` à partir de la position `début`, ou la chaîne vide si les valeurs des paramètres ne le permettent pas ◅ `exemple : sous-chaine('abcdef', 2, 3) renvoie 'cde'`
Tableaux : `tab : tableau[m..n] de type ou tableau de type`
`tab[i]` est l'élément de `tab` situé à la position i.
`Taille(tab : tableau) : entier` renvoie le nombre d'éléments de `tab`
`position_tab(tab : tableau; elt : type ) : entier` renvoie la position de la première occurrence de `elt` dans `tab`, ou -1 si `elt` est absent de `tab`
Expressions Rationnelles: « expression rationnelle » (terme français) ou « expression régulière » (traduction française du terme anglais). On ne rappelle ici que les notions utiles pour les exercices.
Caractères : Tout caractère qui n'est pas un caractère « réservé » se représente lui-même. ***Liste des caractères «réservés» utilisés dans les TD:*** { `.+*?[]- \{}()`}
`.` **: caractère quelconque**
Classes de caractères : représente l'un des caractères de la liste spécifiée entre crochets `[abc]` : l'une des trois lettres 'a' ou 'b' ou 'c' `[a-z]` : un lettre comprise entre 'a' et 'z' `[0-4]` : un chiffre compris entre 0 et 4
Opérateurs multiplicatifs : Ils s'appliquent à l'expression qui les précède et engendrent toutes les chaînes obtenues en répétant la concaténation un certain nombre de fois.
`?` 0 ou 1 fois `*` 0 ou plus `+` 1 ou plus `{m,n}` entre m et n fois
Opérateur de concaténation : il est implicite dès que deux facteurs sont écrits de manière consécutive.
Opérateur de choix : `|` : l'une ou l'autre des expressions en présence.
Priorité décroissante : opérateurs multiplicatifs > concaténation > opérateur de choix
Parenthèses : Utilisées pour modifier les priorités. Par exemple
`ab|c` → 'ab', 'c' `a(b|c)` → 'ab', 'ac'
Traitement des caractères réservés : Deux solutions pour spécifier un caractère réservé à l'intérieur d'une expression régulière :

- le faire précéder du caractère d'échappement \
- le mettre dans une classe de caractères –

exemples: `.` représente un caractère quelconque alors que `\.` et `[.]` représentent le caractère '.'