Algorithmique – 1

Bertrand LIAUDET

SOMMAIRE

Première édition : octobre 2007
Deuxième édition : novembre 2008
Troisième édition : janvier 2009

INTRODUCTION

1. Algorithme et algorithmique en général

Algorithmique

Nom commun : science des algorithmes. Etude des objets et des méthodes permettant de produire des algorithmes.

Adjectif : relatif aux algorithmes.

Algorithme

Définition 1

Texte décrivant sans ambiguïté une méthode pour résoudre un problème.

Définition 2

Texte décrivant sans ambiguïté une suite ordonnée des actions (ou opérations ou traitements) à effectuer pour arriver au(x) résultat(s) attendu(s). En général, certains éléments sont fournis au départ pour effectuer les opérations.

Bilan

Un algorithme est une recette !

Domaine d'application

Un algorithme peut s'appliquer à n'importe quel domaine.

Mathématique

Algorithme du calcul du PGCD de deux nombres (Euclide au $3^{ème}$ siècle avant JC).

Eléments de départ : 2 entier

Résultat : le PGCD

Opérations : des opérations mathématiques.

Informatique

Algorithme d'affichage d'un fond d'écran animé.

Elément de départ : aucun.

Résultat : l'affichage du fond d'écran.

Opération : des opérations mathématiques et informatiques.

Cuisine

Algorithme (recette) du gâteau au chocolat

Recette du gâteau au chocolat
Début
Séparer le blanc des jaunes.
Mélanger le sucre aux jaunes.
Etc.
Faire cuire
Fin

Eléments de départ : les ingrédients.

Résultat : le gâteau.

Opérations : les opérations de cuisine.

Le document de montage d'un meuble en kit

Eléments de départ : les planches, les vis, etc.

Résultat : le meuble (une armoire, par exemple).

Opérations : les opérations de montage.

REMARQUE :

C'est plus ou moins bien fait ! Ca vaut pour la description d'une recette comme pour les algorithmes informatiques !

Notion d'entrée, de sortie et d'instruction

Les actions décrites dans un algorithmes manipulent des objets fournis au départ pour permettre de produire le ou les résultats.

- Les ENTREES sont les éléments de départ.
- Les SORTIES sont les résultats produits.
- Les INSTRUCTIONS sont les actions qui sont décrites dans l'algorithme.

Distinction entre les types d'algorithme

Trois éléments distinguent les différents types d'algorithme :

- Le type des SORTIES : produire un gâteau ou produire un nombr, par exemple.
- Le type des ENTREES : des œufs ou des tableaux de nombres, par exemple.
- Le type des INSTRUCTIONS : mélanger les œufs et le sucre ou comparer la valeur de deux données.

2. Algorithme informatique

Informatique

L'informatique est la science du traitement automatique de l'information. Les anglo-saxons parlent de computeur science.

Est informatique ce qui est relatif à cette science.

Algorithme informatique

Les algorithmes informatiques sont ceux qui décrivent des traitements automatisées d'informations qui ont lieu dans des ordinateurs.

Ecran et clavier

L'écran et le clavier d'un ordinateurs sont les deux périphériques fondamentaux qui permettront de fournir des entrées et d'afficher les résultats.

3. Langages algorithmiques

Les 3 sortes de langages algorithmiques

- Les langages textuels.
- Les langages graphiques (organigrammes, diagramme d'activités).
- Les langages symboliques.

Ces trois sortes de langage ont les mêmes objectifs et ont la même sémantique. La différence se situe uniquement au niveau de la présentation (graphie et syntaxe).

Les langages textuels (pseudo-code)

Ce sont des langages qui décrivent l'algorithme :

- avec un langage alphabétique
- en écrivant les instructions les unes en dessous des autres
- en utilisant un principe d'indentation.

De ce fait, il y a une représentation spatiale à deux dimensions de l'algorithme.

Les langages textuels sont les langages utilisés le plus communément pour écrire des algorithmes : on parle aussi de **pseudo-code**.

Les langages graphiques (organigrammes, diagramme d'activités)

Ce sont des langages qui utilisent des symboles graphiques à deux dimensions pour écrire l'algorithme.

On retrouve les organigrammes avec les diagrammes d'activités d'UML.

Les langages graphiques (organigrammes) sont rarement utilisés pour les algorithmes informatiques « de bas niveau » (type procédure ou fonction). On les utilise généralement pour des descriptions plus générales (par exemple pour décrire le déroulement d'un cas d'utilisation avec ses différents scénarios).

Les langages symboliques

Ce sont des langages qui décrivent l'algorithme :

- avec un langage symbolique
- en écrivant les instructions les unes à la suite des autres sur la même ligne
- donc, sans utiliser de principe d'indentation.

Les langages symboliques ne sont utilisés que par quelques théoriciens. Leur principal intérêt est de pouvoir écrire très rapidement des algorithmes sur un ticket de métro !

Différents approches du pseudo-code

Il y a plusieurs approche dans l'usage du pseudo-code :

- Pseudo-code avec déclaration ou non des variables locales.
- Pseudo-code avec déclaration ou non des types.
- Pseudo-code avec usage ou non de débranchements structurés.
- Pseudo-code avec duplication ou non des début de bloc
- Pseudo-code avec marquage des blocs ou non (numérotation par exemple)

Le choix de ce cours suivra le principe suivant : RIEN DE TROP ! L'écriture algorithmique se limitera au strict nécessaire.

Ce qui conduit à choisir les options suivantes :

- Autant que possible, pas de déclaration des variables locales.
- Autant que possible, pas de déclaration des types grâce à l'utilisation d'une convention de nommage en fonction des types).
- Usage de débranchements structurés.
- Pas de duplication des débuts de bloc.
- Autant que possible, pas de marquage des blocs.

4. Contenu pédagogique

1. **Algorithmique et langage algorithmique**.
2. **Notions fondamentales**
   - Lire, Ecrire,
   - Variable, Type, Affectation (assignation),

- Expression, Evaluation,
- Test, Boucle, Débranchement,
- Bloc, Indentation, Imbrication,
- Procédure, Fonction, Entrée, Sortie, Entrée-Sortie,
- Simulation,
- Complexité, Optimisation.

NOTIONS FONDAMENTALES

1. Programme, afficher, instruction, bloc, indentation

Premier exemple

```
Programme AfficherBonjour
    Afficher (« Bonjour ») ;
Finprog
```

Mots-clés

Les mots-clés sont les mots constant du langage algorithmique. Dans l'exemple proposé, ils sont soulignés. Dans la pratique, on ne les souligne pas. On a ici 4 mots-clés : Programme, Début, Afficher et Fin.

Programme

Un programme est un algorithme. On lui donne un nom (« AfficherBonjour «)

Afficher

Il n'y a qu'une seule instruction : l'instruction Afficher (ou Ecrire).

Afficher est une instruction du langage algorithmique. C'est l'instruction qui permet d'afficher quelque chose à l'écran.

Instruction

« Afficher » est une instruction. En général, on termine les instructions par un point-virgule. C'est toutefois facultatif.

Notion de bloc

L'algorithme commence avec « Programme» et finit avec « Finprog». Tout ce qui se trouve entre « Programme » et « Finprog » constitue un bloc d'instructions, c'est-à-dire un ensemble d'instructions. « Programme » marque le début du bloc. « Finprog » marque la fin du bloc.

Indentation : question de présentation

Le bloc d'instructions est décalé (d'une tabulation) par rapport au début du bloc qui la contient. La fin du bloc est mise juste en dessous du début du bloc.

Programme AfficherBonjour

La première ligne consiste à donner un nom au programme juste après le mot-clé : programme.

Forme générale d'un programme

Programme *NomDuProgramme*

Instructions ;

Fin

2. Lire, variable, affectation, expression, évaluation

Deuxième exemple

```
Programme Fahrenheit
Début
    Lire (Celcius)
    Fahr ← Celcius * 9 / 5 + 32
    Afficher (Fahr) ;
Fin
```

Variables

On a deux variables dans l'algorithme : Fahr et Celcius.

Affectation

« fahr ← 9 / 5 celcius + 32 ; » est une affectation.

Une affectation comporte trois parties :

- A gauche : le nom d'une variable
- Au milleu : le symbole d'affectation : « ← »
- A droite : une expression mathématique à évaluer

Une affectation est une instruction. C'est l'instruction de base de toute la programmation.

L'affectation est l'instruction qui permet de donner la valeur de l'expression de droite à la variable de gauche (appelée parfois « left value »).

Expression et évaluation

Les algorithmes contiennent souvent des expressions mathématiques. Ici : « Celcius * 9 / 5 + 3 »

Ces expressions sont évaluées selon les règles d'évaluation classiques des mathématiques. Leur évaluation fournit une valeur qui sera utilisée dans l'instruction où l'expression se trouve.

Lire

La fonction de lecture permet de récupérer la valeur d'une variable

A cette fonction, on passe la ou les variables à lire.

Les valeurs saisies au clavier sont affectées dans les variables de la fonction lecture.

```
Lire (Celcius)
```

Est équivalent à :

```
Celcius ← Lire ( )
```

Est équivalent à :

```
Celcius ← Clavier
```

On affecte le clavier (ce qui est saisi au clavier) dans la variable Celcius.

On peut passer plusieurs paramètre à la fonction de lecture :

```
Lire (a, b, c)
```

Afficher

Pour afficher la valeur d'une variable, on la passe en paramètre à la fonction afficher. A noter qu'il n'y a plus de guillemets : on utilisait les guillemets pour distinguer les textes qu'on veut afficher des variables qu'on veut aussi afficher.

```
Afficher (Fahr) ;
```

Est équivalent à :

```
Ecran ← Fahr
```

On affecte l'écran (on affiche à l'écran) la valeur de la variable Fahr.

Forme générale d'un programme

```
Programme
        Lecture
        Traitement
        Affichage
Fin
```

Cette forme est très importante à retenir.

Circulation de l'information

Un programme consiste en une circulation d'information.

Notre algorithme peut être décrit ainsi :

- L'utilisateur choisit une valeur dans sa tête
- Cette information est transmise à ses mains
- Ses mains transmettent l'information au clavier
- Le clavier transmet l'information à la variable « Celcius »
- La variable « Celcius » transmet l'information à la variable « Fahr »
- La variable « Fahr » transmet l'information à l'écran
- L'écran transmet l'information aux yeux de l'utilisateur
- Les yeux de l'utilisateur transmettent l'information au cerveau de l'utilisateur.

Retour sur la notion d'instruction

Toute instruction peut se ramener à une ou plusieurs affectations.

- instructions-affectations qui permettent de récupérer de l'information de l'extérieur : « lire ».
- instructions-affectations qui permettent d'envoyer de l'information à l'extérieur : « écrire ».
- instructions-affectations qui permettent de modifier les informations dans le programme : « affectation. »

Bilan du vocabulaire d'un algorithme

Dans un algorithme on trouve :

- Des mots-clés (noms fixes) : Programme, Afficher, Lire, ←
- Des noms variables : Fahrenheit, Celcius, Fahr, etc.
- Des valeurs : 9, 32, « bonjour », etc.
- Des expressions : 9 / 5 celcius + 32, etc. Une expression est constituée de valeurs, de variables et d'opérateurs (+, /, etc.).

3. Variables, types et expression

Distinction entre variable mathématique et variable informatique

En mathématique, la variable est un symbole, généralement une lettre, défini de telle sorte que ce symbole peut être remplacé par **0, 1 ou plusieurs valeurs**. Les valeurs possibles d'une variable dans une situation donnée (x dans une équation) ne sont **pas forcément connues**. Les exercices d'algèbres consistent souvent à mettre au jour les valeurs possibles pour les variables. L'ensemble des valeurs possible pour une variable n'est **pas modifiable**.

En informatique, une variable a toujours **1 et 1 seule valeur**. Cette valeur est toujours **connue**. Cette valeur **peut être modifiée**.

Description d'une variable informatique

Caractéristiques d'une variable

- un nom : celcius : c'est l'identificateur de la variable. Le nom peut faire référence à la valeur ou au contenant selon son usage.
- une valeur (ou contenu): 30 : c'est la traduction en fonction du type de l'état physique réel de la variable.
- un contenant : c'est le bout de mémoire dans lequel la valeur est stockée. Il est symbolise par un carré.
- une adresse : ad_celsius, c'est la localisation du contenant.
- un type : Il permet de définir l'ensemble des valeurs possibles pour la variable (entier, réel, caractère, booléen, etc.).
- une signification (ou sens) : par exemple, celcius c'est la température en °C. La signification est conventionnelle (c'est le programmeur qui la choisit). Mais elle est très importante.

Représentation schématique des variables

Il faut se représenter clairement ce qu'est une variable C'est quelque chose où je peux mettre une information (un tiroir, une page blanche, une ardoise, un contenant). Cette chose est fixe : elle a une position géographique (c'est son adresse : ad_n) et un nom (n) qui me facilite la vie (c'est pour cela qu'il vaut mieux qu'il soit significatif, sinon on pourrait la nommer par son adresse!). Le nom, l'adresse et le contenant ne sont pas modifiables.

Une variable a aussi un contenu : c'est sa valeur, c'est-à-dire l'information qu'elle contient. On peut modifier la valeur. Une variable a toujours un contenu.

Du bon usage : bien nommer les variables

Il faut nommer significativement les variables !

En effet, dans le programme précédent, on pourrait appeler fahr et celcius x et y, ou même celcius et fahr ! Comme ça, on aurait toutes les chances de ne plus rien y comprendre ! Pensez à la relecture!

Double usage du nom d'une variable

Le nom d'une variable fait référence soit la valeur, soit le contenant.

- Quand le **nom** "celcius" est utilisé dans une expression à évaluer, "celcius" c'est la **valeur** de "celcius". "celcius" est utilisé en entrée de l'affectation : il est utilisé mais pas modifié.
- Quand le **nom** fahr est utilisé à gauche de l'affectation. fahr c'est le **contenant** de fahr. On parle aussi de "lvalue" ("left value", "valeur à gauche" de l'affectation, ou "valeur_g"). fahr est utilisé en

Les 4 + 1 types élémentaires

Le type permet de définir l'ensemble des valeurs possibles pour la variable.

Il y a **4 types élémentaire :**

- Entier
- Réel
- Caractère
- Booléen

Auxquels on peut ajouter **un pseudo type élémentaire** :

- Chaîne de caractères

Type et convention algorithmique

Dans les algorithmes, on précisera le type que quand ce sera nécessaire.

On choisira les noms de variables de telle sorte qu'ils correspondent au type.

Voici les noms habituellement associés aux types élémentaires :

- Entier : i, j, k, n, m
- Réel: x, y, z
- Caractère : c, c1, c2
- Chaîne de caractères : ch, ch1, ch2
- Booléen : flag, fg, fg1, drapeau, dp, dp1, bool
  et les symboles ◁ pour un drapeau à vrai et ▷ pour un drapeau à faux.

Expression et évaluation

Constituants d'une expression

Une expression est constituée par :

- Des constantes littérales : 3, -4.5, ‘C’, « bonjour »
- Des nom de variables
- Des opérateurs
- Des noms de fonctions
- Des parenthèses

Valeur et type d’une expression

Une expression à une valeur qui est d’un certain type parmi les type élémentaire.

Evaluation d’une expression

Pour trouver la valeur d’une expression, on évalue l’expression, ce qui consiste à faire les opérations contenue dans l’expression en suivant l’ordre de priorité des opérateurs, des fonctions et des parenthèses.

Exercices – Série 1 - Affectation

Exercice 1

Ecrire programme qui calcule le double d'un entier.

Exercice 2

Écrire un programme qui transforme des degrés Celsius en degrés Fahrenheit. Le zéro Celsius correspond à 32 °F; 100 °C égale 212 °F.

Exercice 3

Écrire un programme qui calcule le prix TTC.

Exercice 4

Écrire un programme qui calcule la circonférence d'un cercle et l'aire du disque délimité par ce cercle.

Exercice 5

Trouvez ce que fait le programme suivant grâce à un exemple, puis prouvez-le.

```
Programme exo4
    lire(a, b)
    a ← b-a
    b ← b-a
    a ← a+b
    afficher (a, b)
Fin
```

Exercice 6

Trouvez ce que fait le programme suivant grâce à un exemple, puis prouvez-le.

```
Programme exo4
    lire(a, b, c)
    a ← a + b + c ;
    b ← b + c ;
    c ← a - c ;
    a ← a - c ;
    b ← c - b + a;
    c ← c - b;
    afficher (a, b, c)
Fin
```

4. Test

Troisième exemple

```
Programme RacineCarré
        Lire (x)
        si x  < 0 alors
                Afficher(« Pas de racine »)
        sinon
                racine = racineCarrée(x)
                Afficher (racine)
        finsi
Fin
```

si – sinon – finsi

Pour pouvoir effectuer une instruction sous condition, on utilise le « si ».

Si une condition est réalisée, alors, on exécute la série d'instructions du bloc « si ». Le début du bloc « si » est marqué par le « si », la fin du bloc « si » est marquée par le « sinon », ou bien, s'il n'y a pas de « sinon », par le « finsi ».

Le sinon correspond à l'alternative du si : ici le cas sinon correspond à x >= 0.

Le finsi vient fermer le bloc ouvert par le sinon.

Evaluation de la condition

La condition (x <0 dans l'exemple) est une expression de type booléen qui est évaluée. Elle vaut soit VRAI, soit FAUX. Si elle vaut vrai, c'est le bloc « si » qui sera exécuté. Si elle vaut faux, c'est le bloc « sinon ». A noter que FAUX est équivalent à 0 et VRAI à tout sauf 0.

Bloc d'instructions

Un « si » ouvre un bloc d'instructions : les instructions dans le bloc sont donc décalées d'une tabulation par rapport au « si ».

Le « sinon » ferme le bloc d'instructions du « si » : il est donc placé sous le « si ».

Le « sinon » ouvre aussi un nouveau bloc d'instructions : les instructions du cas « sinon » sont donc décalées d'une tabulation par rapport au « sinon ».

Le « finsi » ferme le bloc d'instruction du « sinon » ou celui du « si » quand il n'y a pas de « sinon ». Il est donc placé sous le « sinon » ou sous le « si ».

Principe d'imbrication

Dans un bloc instruction, on peut mettre n'importe quel type d'instruction. On peut donc imbriquer les instructions les unes dans les autres, les tests dans les tests, dans les boucles, etc.

En cas d'imbrications multiples et d'algorithme long, on peut numéroter les débuts et fin des blocs d'instructions dans une logique de paragraphe : B1, B1.1, B1.2, B1.1.1, etc.

On peut aussi tirer un trait entre le début et la fin du bloc.

Fonction racine

Dans l'exercice, il ne s'agit pas de trouver une méthode pour calculer la racine carrée. On peut donc utiliser la fonction racineCarrée. C'est un principe général. On peut se donner toutes les fonctions mathématiques dont on a besoin, sauf quand l'exercice précise qu'il faut la fabriquer.

Remarque sur l'écriture algébrique dans les algorithmes

Dans les algorithmes, on n'utilise jamais de barres de fraction, ni de symboles comme racine carrée, carré, ou autre.

Les expressions algébriques doivent être formées avec les 4 opérateurs de base : +, -, *, / et avec les parenthèses en respectant l'ordre de priorité des opérateurs, et avec des fonctions du genre de racineCarrée(x), carré(x), puissance(x, n), log(x), ln(x), factoriel(n), etc.

Remarque sur la structure générale du programme

On retrouve le « lire » en premier.

Ensuite, traitements et affichage sont un peu mixés.

Malgré cela, la structure : « lire, traiter, afficher » reste pertinente.

Tous les tests

si – sinon - finsi

```
si condition alors
        bloc1
sinon
        bloc2
finsi
```

Le sinon est facultatif :

```
si condition alors
        bloc1
finsi
```

si – sinonsi – etc. – sinon - finsi

```
si condition alors
        bloc1
sinonsi
        bloc2
sinonsi
```

```
        bloc3
…
sinon
        bloc4
finsi
```

Le sinon est facultatif.

Cas où – cas1 - cas2 … - défaut - fincas

```
cas où variable vaut
cas 1 : expression ou liste d'expression
        bloc1
cas 2 : expression ou liste d'expression
        bloc2
…
défaut
        blocDefaut
finsi
```

L'instruction « cas où » fait une comparaison d'égalité entre la variable et l'expression ou une liste d'expression.

Le défaut est facultatif.

Méthode d'analyse : l'arbre dichotomique

Toute situation avec des tests pourra être analysé en construisant un arbre de décision dichotomique.

Exercices – Série 2 - Tests

Exercice 1

Écrire un programme qui calcule la valeur absolue de la différence de deux nombres.

Exercice 2

Écrire un programme qui dit si un entier est pair ou pas.

Exercice 3

Écrire un programme qui calcule le maximum de deux nombres.

Exercice 4

Écrire un programme qui calcule le maximum de trois nombres.

Exercice 5

Écrire un programme qui inverse les valeurs de deux variables

Exercice 6

Écrire un programme qui permet de redéfinir 3 variables, X, Y, Z de façon à obtenir : X < Y < Z

Exercice 7

Ecrire une fonction qui permet de calculer le prix d'un certain nombre de photocopies en appliquant le tarif suivant : les 10 premières coûtent 0,10 E pièce, les 20 suivantes 0,08 E pièce et toutes les autres 0,07 E pièce.

Exercice 8

Écrire un programme qui résout dans IR une équation du premier degré en traitant tous les cas possibles (si a=0, etc.)

Exercice 9

Écrire un programme qui résout dans IR une équation du second degré en traitant tous les cas possibles (si a=0, etc.)

5. Boucle

Quatrième exemple

On veut afficher tableau de conversion des fahrenheit en °C pour des fahrenheit allant de 0 à 300 de 20 en 20.

```
Programme listeFahr
    Fahr ← 0
    Tant que fahr < = 300
        Celcius ←  5 * (fahr – 32) / 9
        Afficher (fahr, celcius)
        Fahr ← fahr + 20
    Fin tant que
Fin
```

Boucle tant que

La boucle tant que ouvre un bloc d'instructions.

Ce bloc d'instructions est répété tant que la condition est vérifiée.

Les différents types de boucle

Boucle tant que

```
Tant que condition
    Instructions
Fin tant que
```

Boucle avec tant que final

```
Répéter
    Instructions
Tant que condition
```

Boucle jusqu'à ce que

```
Répéter
    Instructions
Jusqu'à ce que condition
```

Boucle avec compteur : pour i de 1 à n par pas de 1

```
Pour i de 1 à N / pas
```

```
    Instructions
Fin pour
```

Boucle sans fin

```
Boucle
    Instructions
Fin de boucle
```

Les débranchements de boucle : débranchements structurés

Break

Le break permet de quitter la boucle.

Next

Le suivant permet de sauter à la fin de la boucle mais sans quitter la boucle.

Le goto : débranchement non-structuré : INTERDIT !!!

Le GOTO

Le GOTO est une rupture de séquence qui permet de sauter le déroulement continu des instructions (la séquence) en allant directement à la borne précisée par le goto.

Le GOTO permet d'aller n'importe où : on n'utilisera jamais le GOTO.

CF : « GO TO statement considered harmful » de Dijkstra (1968).

Notion de rupture de séquence

Les tests et les boucles sont des ruptures de séquences : les instructions ne vont pas s'exécuter l'une après l'autre.

Le test fait sauter en avant.

La boucle fait sauter en arrière, et éventuellement en avant.

Le test et la boucle sont des GOTO structurés.

Traduction d'une boucle tant que avec un GOTO structuré

```
Programme listeFahr
    Fahr ← 0
    borne tantque
    si fahr > 300
        goto borne fintantque
    finsi
    Celcius ← 5 * (fahr – 32) / 9
```

```
        Afficher (fahr, celcius)
        Fahr ← fahr + 20
        goto borne tantque
        borne fintantque
Fin
```

Notion de rupture de séquence

Les débranchements sont des ruptures de séquences.

Exercices – Série 3 - Boucle

Exercice 1

Écrire un programme qui affiche la table de multiplication par 7.

Exercice 2

Écrire un programme qui calcule la somme des n premiers nombres (on utilisera une boucle).

Écrire une variante qui calcule la somme des n premiers nombres pairs.

Exercice 3

Réécrire le programme précédent en effaçant l'écran au début et en donnant la possibilité à l'utilisateur de recommencer le programme ou de quitter le programme.

Exercice 4

Écrire un programme qui calcule X puissance n, avec n entier relatif. On traitera tous les cas particuliers.

Exercice 5

Ecrire un programme pour déterminer si un nombre est premier.

Rappel : un nombre premier n'a pas de diviseurs.

Exercice 6

Ecrire un programme qui permet de calculer factorielle N (N !).

Rappels : 0 ! =1, N ! = 1*2*3*…*N

Exercice 7

On désire calculer le terme d'ordre n de la suite suivante définie par :

S(0)=1, S(n)=*2F(n-1)+1

Écrire un programme qui permet de calculer F(n).

Exercice 8

On désire calculer la racine carrée par la méthode de Newton.

Racine(A) est donnée par le calcul de la suite suivante :

S(0)=0, S(n+1) = 0,5 * (S(n) + A / S(n))

Quand deux valeurs successives de la suite sont identiques, on a trouvé la racine.

Écrire un programme qui permet de calculer racine(A) avec cette méthode.

Exercice 9

On désire calculer le terme d'ordre n de la suite de Fibonacci définie par :

F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2).

Écrire un programme qui permet de calculer F(n).

Exercice 10

Ecrire un programme qui permet de calculer le développement limité de sin x.

$\sin(x) = x - x^3 / 3! + x^5 / 5 ! - x^7 / 7 !$ etc.

La condition d'arrêt du calcul sera que le terme $x^n/n !$ soit plus petit qu'une constante EPSILON donnée. (Epsilon est la plus petite valeur possible pour un réel).

Exercice 11

Un nombre est dit parfait s'il est égal à la somme de ses diviseurs excepté lui même. Par exemple, 6 est parfait, 6=1+2+3. 28 est parfait : 28=1+2+4+7+14

Ecrire un programme qui permet d'afficher tous les nombres parfaits plus petit qu'une valeur N.

Exercice 12

Un triplet d'entiers naturels (x, y, z) est dit pythagoricien si et seulement si on a : x2 +y2=z2.

Ecrire un programme qui, pour une valeur de n donnée, affiche tous les triplets pythagoriciens tels que x2 + y2 <=n.

Méthode d'analyse

Dès qu'un traitement doit être répété, on fera appel à une boucle.

La boucle de base est la boucle tant que. Toute répétition doit donc être traitée en première hypothèse par un tant que.

Si on sait combien de répétition on doit effectuer, on aura intérêt à utiliser une boucle « pour ».

Dans le cas d'un « tant que », si la première occurrence du traitement est nécessairement effectuée, on aura intérêt à utiliser un « répéter ».

6. Fonction et procédure

Cinquième exemple

Ecrire une fonction qui calcule la surface d'un rectangle et un programme qui permet de l'utiliser.

```
Fonction surface (larg , long) : réel  //  la surface
// E : larg : réel // la largeur
// E : long : réel // la longueur
Début
        Return  larg * long;
Fin

Programme afficheSurface
        Lire (larg, long)
        Surf = surface(larg, long) ;
        Afficher (surf) ;
Fin
```

La notion de fonction

Entête de la fonction

```
Fonction surface (larg , long) : réel  //  la surface
```

L'en-tête de la fonction permet de décrire le nom de la fonction, le type de son résultat et les variables utilisées par l'algorithme de la fonction.

Corps de la fonction

```
Début
        Return  larg * long;
Fin
```

Le corps de la fonction décrit l'algorithme de la fonction.

L'instruction return permet de renvoyer le résultat de la fonction sur la sortie standad.

Paramètres formels de la fonction

Les variables de l'en-tête de la fonction sont appelés : paramètres formels.

Dans cet exemple, ces paramètres sont passés en entrée. Une variable en entrée est une variable dont on utilise la valeur pour le calcul de l'algorithme mais dont la valeur ne sera pas modifiée.

Sortie standard de la fonction : le propre d'une fonction

```
Fonction surface (larg , long) : réel  //  la surface
```

Une fonction renvoie un résultat sur sa sortie standard. C'est ce qui distingue une fonction d'une procédure.

Le type du résultat fourni par la fonction est précisé dans l'en-tête de la fonction : réel dans l'exemple.

Débranchement de fonction : le return

```
    Return  larg * long;
```

L'instruction return permet de renvoyer le résultat de la fonction sur la sortie standard.

Cette instruction permet aussi de quitter la fonction à tout moment.

Utilisation d'une fonction : les paramètres d'appels de la fonction

```
    Surf = surface(larg, long) ;
```

Quand un programme fait appel à une fonction, les paramètres passés à la fonction sont appelés paramètres d'appel.

Ces paramètres sont des expressions qui sont évaluées. Il peuvent donc être : des constantes littérales (3, 'A', «Bonjour »), des noms de variables ou des expressions. (Quand on a des variables de type complexe, ce sont en général des adresses qui sont passées en paramètre.)

C'est la valeur de l'évaluation qui est passée en paramètre à la fonction (passage par valeur).

Cette valeur est affectée à la variable du paramètre formel correspondant.

La correspondance entre les paramètres d'appels et les paramètres formels se fait par l'ordre des paramètres : le premier paramètre formel prend la valeur du premier paramètre d'appel, et ainsi de suite avec les suivants.

Notion de procédure : paramètres en sortie

Entête et corps

Une procédure est constituée, comme une fonction, d'un en-tête et d'un corps.

Paramètre en sortie

A la différence d'une fonction, une procédure a des paramètres en sortie dans son en-tête.

Ainsi, la fonction précédente pourrait s'écrire :

```
Procédure surface (larg, long, surf)
// E : larg : reel // largeur
// E : larg : reel // longeur
// S : surf : reel // surface
Début
    Surf = larg * long;
Fin
```

On a alors affaire à une procédure.

Auquel cas le programme principal aurait été :

```
Programme afficheSurface
    Lire (larg, long)
    surface(larg, long, surf)
    Afficher (surf)
Fin
```

Une procédure peut avoir plusieurs paramètres en sortie.

Pas de sortie standard

Une procédure, par définition, n'a pas de sortie standard. C'est ce qui la distingue des fonctions.

Paramètres d'appels d'une procédure : passage par adresse

Le principe est le même pour une procédure et pour une fonction pour les paramètres en entrée.

Par contre, pour les paramètres en sortie, le paramètre d'appel ne peut pas être une expression : c'est obligatoirement une variable. On parle alors de passage par référence ou par adresse.

Débranchement de procédure : le return

L'instruction return permet de quitter la procédure à tout moment.

Comme il n'y a pas de sortie standard, le return n'est jamais accompagné d'une expression.

Ecriture « au crayon »

Dans l'algorithme traité, le type des paramètres est évident, leur nom suffit à dire ce qu'ils sont, on peut donc se contenter de préciser le mode de passage (E, ES ou S) qu'on peut représenter avantageusement au crayon par des flèches au dessus des variable :

```
                    ↓   ↓   ↑
Procédure surface (larg, long, surf)
    Surf = larg * long;
Fin
```

Fonction mixte

Une fonction mixte possède une sortie standard et des paramètres en sortie.

Exercices – Série 4

Pour tous les exercices, on aura intérêt à écrire un programme qui teste la procédure ou la fonction écrite.

Exercice 1

Ecrire une procédure ou une fonction qui affiche une table de multiplication

Exercice 2

Ecrire une procédure ou une fonction renvoie le double d'un entier.

Exercice 3

Écrire une procédure ou une fonction qui calcule la circonférence d'un cercle et l'aire du disque délimité par ce cercle.

Exercice 4

Écrire une procédure ou une fonction qui calcule le prix TTC.

Exercice 5

Écrire une procédure ou une fonction qui calcule le maximum de deux nombres.

Exercice 6

Écrire une procédure ou une fonction qui calcule le maximum de trois nombres.

Exercice 7

Ecrire une procédure ou une fonction qui permet de calculer le prix d'un certain nombre de photocopies en appliquant le tarif suivant : les 10 premières coûtent 0,50 F pièce, les 20 suivantes 0,30 F pièce et toutes les autres 0,20 F pièce.

Exercice 8

Écrire une procédure ou une fonction résout dans IR une équation du premier degré en traitant tous les cas possibles (si a=0, etc.)

Exercice 9

Écrire une procédure ou une fonction qui résout dans IR une équation du second degré en traitant tous les cas possibles (si a=0, etc.)

Exercice 10

Ecrire une procédure ou une fonction qui renvoie le nombre de jours d'un mois d'une année donnée.

On rappelle qu'une année est bissextile si le millésime est divisible par 4 et n'est pas divisible par 100, ou si le millésime est divisible par 400.

Exemple : 2000 et 2008 étaient des années bissextiles, 1900 n'était pas une année bissextile.

Exercice 11

Écrire une procédure ou une fonction qui calcule la somme des n premiers nombres (on utilisera une boucle).

Écrire une variante qui calcule la somme des n premiers nombres pairs.

Exercice 12

Écrire une procédure ou une fonction itérative (c'est-à-dire pas récursive) qui calcule X puissance n, avec n entier positif ou nul. On traitera tous les cas particuliers.

Exercice 13

On désire calculer le terme d'ordre n de la suite de Fibonacci définie par :

F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2).

Écrire une procédure ou une fonction itérative (c'est-à-dire pas récursive) qui permet de calculer F(n) et un programme pour la tester.

Exercice 14

Ecrire une procédure ou une fonction pour déterminer si un nombre est premier.

Exercice 15

Ecrire une procédure ou une fonction itérative qui permet de calculer N !

Exercice 16

Ecrire une procédure ou une fonction qui permet de calculer le développement limité de sin x avec une précision Epsilon.

Rappel : $\sin(x) = x - x^3 / 3! + x^5/5 ! \ldots$

Exercice 17

Un nombre est dit parfait s'il est égal à la somme de ses diviseurs propres. Par exemple, 6 est parfait, 6=1+2+3.

1) Ecrire une procédure ou fonction qui dit si un nombre est parfait ou pas.

2) Ecrire une procédure ou une fonction qui, pour une valeur donnée de n, affiche les nombres parfaits < n

3) Ecrire un programme qui utilise cette fonction.

Exercice 18

Ecrire une procédure ou une fonction qui calcule le temps de remontée d'un plongeur à partir d'une profondeur donnée. Le principe de la remontée pour une profondeur inférieure ou égale à 100 m est le suivant :

le plongeur doit faire un palier de 100 secondes à 30 m puis de 30 m à 0, des paliers tous les 3 m, le temps diminuant de 10 secondes à chaque fois.

Les paliers au-dessus de 30 m sont par contre augmentés de 25 secondes tous les 10 m, (ainsi p.ex.: 150 s à 50 m).

La vitesse de remontée est fixée à 1 m/s.

Exercice 19

Ecrire un programme qui permette d'utiliser les fonctions ou procédures des exercices 5, 8 et 11 (max de trois nombres, équation du second degré et calcul de la puissance). Le programme permettra de choisir parmi ces trois possibilité. Il permettra aussi de recommencer ou de choisir d'arrêter. A chaque nouveau choix, on repartira avec un écran effacé en utilisant la procédure effaceEcran( ).

7. Vérification et simulation

Principe

Pour vérifier un algorithme, une méthode consiste à simuler son exécution et à regarder le contenu des variables après chaque instruction.

Mise en œuvre

Pour mettre en œuvre la simulation, il faut faire un tableau. Chaque colonne contient une variable. Chaque ligne correspond à une instruction. On remplit au fur et à mesure la valeur des variables dans le tableau.

Exemple

- ***Algorithme***

```
Programme listeFahr
    Fahr ← 0
    Tant que fahr < = 300
        Celcius ← 5 * (fahr – 32) / 9
        Afficher (fahr, celcius)
        Fahr ← fahr + 20
    Fin tant que
Fin
```

- ***Simulation***

	Fahr	Fahr <=300	Celcius
Fahr ← 0	0		
		Vrai	
Celcius ← 5 * (fahr – 32) / 9			-17,6
Fahr ← fahr + 20	20		
		Vrai	
Celcius ← 5 * (fahr – 32) / 9			-6,7
Fahr ← fahr + 20	40		
		Vrai	
Celcius ← 5 * (fahr – 32) / 9			4,4
	Etc.		
Fahr ← fahr + 20	280		
		Vrai	

Celcius ← 5 * (fahr – 32) / 9			137,8
Fahr ← fahr + 20	300		
		Vrai	
Celcius ← 5 * (fahr – 32) / 9			148,9
Fahr ← fahr + 20	320		
		faux	

8. Complexité et optimisation

Les types de complexité

On distingue 4 types de complexité

La complexité apparente

Elle concerne la difficulté qu'éprouve le lecteur à comprendre l'algorithme. C'est une notion assez subjective. Toutefois, on prendra en compte plusieurs éléments :

- La dénomination significative des variables
- Le bon usage des indentations.
- La modularité de l'algorithme (utilisation de procédures et de fonctions)
- Le niveau d'imbrication. Les débranchements structurés permettent de le limiter.
- La distinction entre l'interface utilisateur et les traitements
- La distinction entre le traitement du cas général et le traitement des cas particuliers
- Le bon usage des commentaires

Cette liste n'est pas exhaustive.

La complexité spatiale

La complexité spatiale correspond à l'encombrement mémoire de l'algorithme.

Elle peut être calculée théoriquement à partir de la connaissance des données mises en œuvre dans l'algorithme.

La complexité temporelle : performance, optimisation

La complexité temporelle correspond au temps d'exécution de l'algorithme.

Elle correspond à l'analyse de la performance.

La complexité temporelle pratique

C'est la mesure réelle du temps d'exécution de l'algorithme.

Elle est calculée en faisant des tests avec différents jeu de données.

Par exemple, on produit les résultats pour les algorithmes de tris selon la taille des tableaux.

Exemple de résultats :

Nombre de lignes :	10	100	1000	10000	50000
BULLE	0,16	20	2400		
EXTRACTION	0,12	7,3	680		
INSERTION	0 ?12	6,7	610		

SHELL	0 ?07	2	37	600	4200
ARBRE	0,2	3,5	50	660	3960
RAPIDE		2	28	365	2140

(d'après C. BAUDOIN et B. MEYER, cité dans GUYOT et VIAL)

La complexité temporelle théorique

C'est un ordre de grandeur théorique du temps d'exécution de l'algorithme.

Ce calcul concerne toujours des algorithmes avec des boucles.

La mesure de la complexité est donnée par : « O ».

« O » signifie : « complexité de l'ordre de ».

On écrit : O(N), ou O(N/2) ou O(N^2) pour une complexité de l'ordre de N, de N/2, de N*N.

- ***Exemple : complexité de la recherche du plus petit diviseur***

Algo 1 :

```
fonctionPPD (N) : entier // plus petit diviseur
    Pour i de 2 à N-1
        Si N mod i = 0
            Return i
        Finsi
    Finpour
    Return 1
FIN
```

Complexité: O(N)

N=100 : O(100)

Algo 2 :

```
fonctionPPD (N) : entier // plus petit diviseur
  si N mod 2 = 0
        return 2
  finsi
  pour i de 3 à racine(N) par pas de 2
        si N mod i = 0
              return i
        finsi
  Finpour
  Return 1
FIN
```

Complexité : O (racine(N)/2)

N=100 : O(5)

- ***Remarques***

On voit dans l'exemple qu'on ne calcule pas la complexité à l'unité près. On n'écrit pas : O(N-2) mais O(N). O n'est qu'une approximation.

Si le temps d'exécution n'est pas fonction de N, la complexité vaut : O (1)

La différentes complexités temporelles

Temps constant : O(1)

Temps linéaire : O(N)

Temps proportionnel : O(N/2)

Temps quadratique : O(N*N)

Temps quadratique amélioré : O(N*N/2)

Temps logarithmique : O(log2(N))

Les types d'optimisation

Corrélativement à la complexité, on peut distinguer 3 types d'optimisation

L'optimisation d'écriture

L'optimisation d'écriture consiste à améliorer tous les points abordées dans la complexité apparente :

- Mieux nommer les variables
- Bien indenter
- Faire des procédures et des fonctions
- Limiter les imbrications par l'usage de débranchements structurés.
- Séparer l'interface utilisateur et les traitements
- Séparer le traitement du cas général et du traitement des cas particuliers
- Commenter intelligemment

L'optimisation temporelle

Elle consiste à choisir des structures de données qui prennent moins de place et à éviter certaines méthodes algorithmique coûteuses en place (comme la récursivité, par exemple).

L'optimisation temporelle

Elle consiste à diminuer la valeur de la complexité « O », ou à faire des tests de performance parmi plusieurs algorithmes en concurrence.