Algorithmique

Alexandre Meslé

Table des matières

Chapitre 1

Notes de cours

1.1 Introduction

1.1.1 Le principe

Exemple 1 - La surprise du chef

Considérons la suite d'instructions suivante :

1. Faites chauffer de l'eau dans une casserole
2. Une fois que l'eau boue, placez les pˆates dans l'eau
3. Attendez dix minutes
4. Versez le tout dans un écumoire
5. Vos pˆates sont prêtes.

Vous l'aurez deviné, il s'agit des grandes lignes de la recette permettant de préparer des pˆates (si vous les voulez al dente, attendez un petit peu moins de 10 minutes). Cette recette ne vous expose pas le détail des réactions chimiques qui font que les pˆates cuisent en dix minutes, ni pourquoi il faut les égoutter. Il s'agit seulement d'une suite d'instructions devant être exécutées a` la lettre. Si vous ne les suivez pas, vous prenez le risque que le résultat ne soit pas celui que vous attendez. Si vous décidez de suivre une recette, vous décidez de vous conformer aux instructions sans poser de questions. Par opposition, vous pouvez décider de créer vous-même une recette, cela vous demandera davantage de réflexion, et vous serez amenéa` élaborer d'une suite d'instructions qui vous permettra de retrouver le même résultat.

Exemple 2 - Ikéa

Considérons comme autre exemple une notice de montage. Elle est composée d'un ensemble d'étapes a` respecter scrupuleusement. Il ne vous est pas demandé de vous interroger sur la validité de ces instructions, on vous demande juste de les suivre. Si vous vous conformez aux indications de la notice, vous parviendrez a` monter votre bibliothèque Louis XV I. Si vous ne suivez pas la notice de montage, il vous restera probablement a` la fin une pièce entre les mains, et vous aurez beau chercher oüla placer, aucun endroit ne conviendra. Vous aurez alors deux solutions: soit vous démontez tout pour reprendre le montage depuis le début, soit vous placez cette pièce dans l'assiette qui est dans l'entrée en attendant le prochain déménagement, et en sachant que la prochaine fois, vous suivrez la notice... Cet exemple est analogue au premier, vous avez entre vos mains une suite d'instructions a` ex´ecuter, si vous les suivez, vous obtenez le résultat attendu, sinon, il y a de très fortes chances que n'obteniez pas le résultat escompté. De la même façon, le but n'est pas que vous vous demandiez pourquoi ou comment ça marche, la notice est faite pour que vous n'ayez pas a` vous poser ce type de question. Si jamais vous décidez de créer un meuble (par exemple, une bibliothèque Nicolas Premier) a` monter soi-même, il vous faudra fournir avec une notice de montage. C'est-à-dire une succession d'étapes que l'acquéreur de ce meuble devra suivre a` la lettre.

Définition

On conclut de de la façon suivante : nous avons vu qu'il existait des s´equences d'instructions faites pour être ex´ecut´ee a` la lettre et sans se poser de questions, c'est le principe de l'algorithme. Nous retiendrons donc que Un algorithme est une séquence d'instructions exécutée de façon logique mais non intelligente.

- Logique parce que la personne (ou la machine) qui ex´ecute les instructions est capable de comprendre et ex´ecuter sans erreur chacune d'elles.
- Non intelligente parce que la personne qui ex´ecute l'algorithme n'est pas suppos´ee apte a` comprendre pourquoi la succession d'´etapes d´ecrite par l'algorithme donne bien un r´esultat correct.

Utilisation en informatique

Les premiers algorithmes remontent a` l'antiquit´e. Par exemple l'algorithme de calcul du plus grand commun diviseur de deux nombres, appel´e maintenant "algorithme d'Euclide". Il s'agissait en g´en´eral de m´ethodes de calcul semblables a` celle que vous utilisez depuis le cours ´el´ementaire pour additionner deux nombres a` plusieurs chiffres. Notez qu'àl'´epoque, on vous demandait juste d'appliquer la m´ethode sans vous tromper, on ne vous a pas expliqu´e pourquoi cette m´ethode marchait a tous les coups. Le principe ´etait donc le même, vous n'aviez pas le niveau en math´ematiques
`
pour comprendre pourquoi la succession d'´etapes qu'on vous donnait ´etait valide, mais vous ´etiez capable d'ex´ecuter chaque ´etape de la m´ethode. Avant même d'avoir dix ans, vous connaissiez donc d´ejàdes algorithmes.

Le mot algorithme prend ´etymologiquement ses racines dans le nom d'un math´ematicien arabe du moyen ˆage: Al-Kawarizmi. Les algorithmes sont extrêmement puissants : en concevant un algorithme, vous pouvez d´ecomposer un calcul compliqu´e en une succession d'´etapes compr´ehensibles, c'est de cette façon qu'on vous a fait faire des divisions (op´eration compliqu´ee) en cours moyen, a` un ˆage o`u votre niveau en math´ematiques ne vous permettait pas de comprendre le fonctionnement d'une division.

Contrairement aux mythe Matrix-Terminator-L'Odyss´ee de l'espace-I, Robot-R2D2 (et j'en passe) un ordinateur fonctionne de la même façon qu'un monteur de bibliothèque (rien a` voir avec l'alpinisme) ou votre cuisinier c´elibataire (il y a quand même des exceptions), il est idiot et pour chaque chose que vous lui demanderez, il faudra lui dire comment faire. Vous aller donc lui donner des successions d'instructions a` suivre, et lui les respectera a` la lettre et sans jamais se tromper. Une suite d'instructions de la sorte est fournie a` l'ordinateur sous la forme de programme. Pour coder un programme, on utilise un langage de programmation, par exemple C, Java, Pascal, VB... Selon le langage utilis´e, une même instruction se code diff´eremment, nous ferons donc dans ce cours abstraction du langage utilis´e. Nous nous int´eresserons uniquement a` la façon de combiner des instructions pour former des programmes, ind´ependamment des langages de programmation. Le but de ce cours est donc de vous apprendre a` cr´eer des algorithmes, c'est-à-dire a` d´ecomposer des calculs compliqu´es en successions d'´etapes simples.

1.1.2 Variables

Une algorithme se pr´esente comme une liste d'instructions, elles sont ´ecrites les unes au dessus des autres et elles sont ex´ecut´ees dans l'ordre, lorsque l'on conçoit une algorithme, il faut toujours avoir en tête le fait que l'ordre des instructions est très important. Le premier concept n´ecessaire pour concevoir un algorithme est celui de variable. Une variable est un emplacement dans la m´emoire o`u est stock´ee une valeur. Une variable porte un nom, ce nom est laiss´e au choix du concepteur de l'algorithme, il doit commencer par une lettre et ne pas comporter d'espace. On se sert du nom d'une variable pour lire la valeur qui s'y trouve ou bien pour la modifier, une variable ne peut contenir qu'une seule valeur a` la fois. Une valeur est `num´erique` s'il s'agit d'un nombre, ou bien `alphanum´erique` s'il s'agit d'une succession de symboles, par exemple des mots. Toute

variable a un type : si une variable est de type numérique, il n'est possible d'y mettre que des valeurs numériques, si une variable est de type alphanumérique, il n'est possible d'y stocker que des valeurs alphanumériques.

L'affectation est une opération permettant de modifier la valeur d'une variable. La syntaxe de l'affectation est la suivante :

```
nomvariable ←– valeur
```

<nomvariable> est le nom de la variable dont on souhaite modifier la valeur, <valeur> est la valeur que l'on veut placer dans la variable. Notez bien que cette valeur doit être de même type que la variable. Par exemple,

```
A ←– 5
```

place la valeur 5 dans la variable A. Si A contenait préalablement une valeur, celle-ci est écrasée. Il est possible d'affecter a` une variable le résultat d'une opération arithmétique.

```
A ←– 5 + 2
```

On peut aussi affecter a` une variable la valeur d'une autre variable

```
A ←– B
A ←– B + 2
```

La première instruction lit la valeur de B et la recopie dans A. la deuxième instruction, donc exécutée après la première, lit la valeur de B, lui additionne 2, et recopie le résultat dans A. Le fait que l'on affecte a` A la valeur de B ne signifie pas que ces deux variables auront dorénavant la même valeur. Cela signifie que la valeur contenue dans B est écrasée par la valeur que contient A au moment de l'affectation. Si la par la suite, la valeur de A est modifiée, alors la valeur de B restera inchangée. Il est possible de faire figurer une variable simultanément a` gauche et a` droite d'une affectation:

```
A ←– A + 1
```

Cette instruction augmente de 1 la valeur contenue dans A, cela s'appelle une incrémentation.

Exemple

Quelles sont les valeurs des variables après l'exécution des instructions suivantes ?

```
A ←– 1
B ←– 2
C ←– 3
D ←– A
A ←– C + 1
B ←– D + C
C ←– D + 1
```

Construisons un tableau nous montrant les valeurs des variables au fil des affectations:

Instruction	A	B	C	D
Début	n.i	n.i	n.i	n.i
A ←– 1	1	n.i	n.i	n.i
B ←– 2	1	2	n.i	n.i
C ←– 3	1	2	3	n.i
D ←– A	1	2	3	1
A ←– C + 1	4	2	3	1
B ←– D + C	4	4	3	1
C ←– D + 1	4	4	2	1

n.i signifie ici non initialisée. Une variable est non initialis´ee si aucune valeur ne lui a ´et´e explicitement affect´ee. A ←− 1 modifie la valeur contenue dans la variable A. A ce moment là de l'ex´ecution, les valeurs des autres variables sont inchang´ees. B ←− 2 modifie la valeur de B,

les deux variables A et B sont maintenant initialis´ees. C ←− 3 et D ←− A initialisent les deux variables C et D, maintenant toutes les variables sont initialis´ees. Vous remarquerez que D a ´et´e initialis´ee avec la valeur de A, comme A est une variable initialis´ee, cela a un sens. Par contre, si l'on avait affect´e a` D le contenu d'une variable non initialis´ee, nous aurions ex´ecut´e une instruction qui n'a pas de sens. Vous noterez donc qu'il est interdit de faire figurer du cotédroit d'une affectation une variable non initialisée. Vous remarquez que l'instruction D ←− A affecte a` D la valeur de A, et que l'affectation A ←− C + 1 n'a pas de cons´equence sur la variable D. Les deux variables A et D correspondent a` deux emplacements distincts de la mémoire, modifier l'une n'affecte pas l'autre.

1.1.3 Littéraux

Un littéral est la repr´esentation de la valeur d'une variable. Il s'agit de la façon dont on ´ecrit les valeurs des variables directement dans l'algorithme.

– num´erique : 1, 2, 0, −4,...
– alphanum´erique : "toto", "toto01", "04", ...

Attention, 01 et 1 repr´esentent les mêmes valeurs num´eriques, alors que "01" et "1" sont des valeurs alphanum´eriques distinctes. Nous avons vu dans l'exemple pr´ec´edent des litt´eraux de type num´erique, dans la section suivante il y a un exemple d'utilisation d'un variable de type alphanum´erique.

1.1.4 Convention d'écriture

Afin que nous soyons certains de bien nous comprendre lorsque nous r´edigeons des algorithmes, d´efinissons de façon pr´ecise des règles d'´ecriture. Un algorithme s'´ecrit en trois parties :

1. Le titre, tout algorithme porte un titre. Choisissez un titre qui permet de comprendre ce que fait l'algorithme.
2. Les déclarations de variables, vous pr´eciserez dans cette partie quels noms vous avez d´ecid´e de donner a` vos variables et de quel type est chacune d'elle.
3. Les instructions, aussi appel´e le corps de l'algorithme, cette partie contient notre succession d'instructions.

Par exemple,

```
Algorithme : Meanless
variables:
num´eriques : A, B, C
alphanum´eriques : t
DEBUT
    A ←− 1
    B ←− A + 1
    C ←− A
    A ←− A + 1
    t ←−"this algorithm is dumb"
FIN
```

La lisibilit´e des algorithmes est un critère de qualit´e pr´epond´erant. Un algorithme correct mais ind´echiffrable est aussi efficace qu'un algorithme faux. Donc c'est un algorithme faux. Vous devrez par cons´equent soigner particulièrement vos algorithmes, ils doivent être faciles a` lire, et r´edig´es de sorte qu'un lecteur soit non seulement capable de l'ex´ecuter, mais aussi capable de le comprendre rapidement.

1.1.5 Entrées-sorties

De nombreux algorithmes ont pour but de communiquer avec un utilisateur, cela se fait dans les deux sens, les sorties sont des envois de messages a` l'utilisateur, les entrées sont des informations fournies par l'utilisateur.

Saisie

Il est possible de demander a` un utilisateur du programme de saisir une valeur. La syntaxe de la saisie est la suivante :

```
Saisir < nomvariable >
```

La saisie interrompt le programme jusqu'àce que l'utilisateur ait saisi une valeur au clavier. Une fois cela fait, la valeur saisie est placée dans la variable nomvariable. Il est possible de saisir plusieurs variables a` la suite,

```
Saisir A, B, C
```

place trois valeurs saisies par l'utilisateur dans les variables `A`, `B` et `C`.

Affichage

Pour afficher un message a` destination de l'utilisateur, on se sert de la commande

```
Afficher < message>
```

Cette instruction affiche le `<message>` a` l'utilisateur. Par exemple,

```
Afficher "Hello World"
```

affiche "Hello World" (les guillemets sont très importantes !). Il est aussi possible d'afficher le contenu d'une variable,

```
Afficher A
```

affiche l'écran le contenu de la variable `A`. On peut entremêler les messages et les valeurs des variables. Par exemple, les instructions

```
Afficher "La valeur de la variable A est";
Afficher A;
```

ont le même effet que l'instruction

```
Afficher "La valeur de la variable A est", A
```

Lorsque l'on combine messages et variables dans les instruction d'affichage, on les sépare par des virgules. Notez bien que ce qui est délimitépar des guillemets est affichétel quel, alors tout ce qui n'est pas délimitépar des guillemets est considérécomme des variables.

Exemple

Cet algorithme demande a` l'utilisateur de saisir une valeur numérique, ensuite il affiche la valeur saisie puis la même valeur incrémentée de 1.

```
Algorithme : Affichage incrément

Algorithme : Valeurs Distinctes

variables:
numériques : a, b
DEBUT
    Afficher "Saisissez une valeur numérique"
    Saisir a
    b ←— a + 1
    Afficher "Vous avez saisi la valeur ", a, "."
    Afficher a, "+ 1 = ", b
FIN
```

1.2 Traitements conditionnels

On appelle traitement conditionnel un bloc d'instructions dont l'exécution est soumise a` la vérification d'un test.

1.2.1 SI ... ALORS

La syntaxe d'un traitement conditionnel est la suivante :

```
si < condition> alors
   < instructions>
fin
```

Les <instructions> ne sont exécutées que si <condition> est vérifiée. Par exemple,

```
si A = 0 alors
                                    Afficher "La valeur de la variable A est nulle."
fin
```

Si la variable A, au moment du test, a une valeur nulle, alors l'instruction `Afficher "La valeur de la variable A est nulle."` est exécutée, sinon, elle est ignorée.

Conditions

Une condition peut être tout type de test. Par exemple,

```
A = 2
A = B
B <> 7
2 > 7
```

La condition `A = 2` est vérifiée si la valeur contenue dans A est 2. `A = B` est vérifiée si les valeurs contenues dans A et dans B sont les mêmes. `B <> 7` est vérifiée si B contient une valeur différente de 7. `2 > 7` est vérifiée si 2 est supérieur a` 7, donc jamais, cette condition est donc fausse et ne dépend pas des valeurs des variables.

Si étendu

Le traitement conditionnel peut être étendue de la sorte :

```
si < condition> alors
   < instructions>
sinon
                                    < autresinstructions >
fin
```

Si <condition> est vérifiée, les <instructions> sont exécutées. Dans le cas contraire, donc si <condition> n'est pas vérifiée, alors ce sont les <autresinstructions> qui sont exécutées. Par exemple,

```
Algorithme : Valeurs Distinctes
variables: num´eriques : a, b
D E B U T
    Afficher ”Saisissez deux valeurs numériques”
    Saisir a,b
    si a = b alors
        Afficher ”Vous avez saisi deux fois la même valeur, a` savoir ”, a, ”.” sinon
                                    Afficher ”Vous avez saisi deux valeurs différentes, ”, a, ” et ”, b, ”.”
                                                                                                    fin
FIN
```

Dans l'exemple ci-dessus, la condition `a = b` est ´evalu´ee. Si a` ce moment-làles variables a et b contiennent la même valeur, alors la condition `a = b` sera v´erifi´ee. Dans ce cas, l'instruction `Afficher "Vous avez saisi deux fois la même valeur, a` savoir ", a, "."` sera ex´ecut´ee. Si la condition `a = b` n'est pas v´erifi´ee, donc si les variables a et b ne contiennent pas la même valeur au moment de l'´evaluation de la condition, c'est alors l'instruction `Afficher "Vous avez saisi deux valeurs différentes, ", a, " et ", b, "."` qui sera ex´ecut´ee.

Imbrication

Il est possible d'imbriquer les `SI` a` volont´e :

```
si a < 0 alors
    si b < 0 alors
        Afficher ”a et b sont négatifs” sinon
                                                    Afficher ”a est négatif, b est positif”
    fin
sinon
    si b < 0 alors
        Afficher ”b est négatif, a est positif” sinon
        Afficher ”a et b sont positifs”
fin
fin
```

Si par exemple a et b sont tous deux positifs, alors aucun des deux tests ne sera v´erifi´e, et c'est donc le `sinon` du `sinon` qui sera ex´ecut´e, a` savoir `Afficher "a et b sont positifs"`.

Connecteurs logiques

Les connecteurs logiques permettent de d'´evaluer des conditions plus complexes. Deux sont disponibles :

– et : la condition `<condition1> et <condition2>` est v´erifi´ee si les deux conditions `<condition1>` et `<condition2>` sont v´erifi´ees simultan´ement.

ou : la condition `<condition1> ou <condition2>` est v´erifi´e si au moins une des deux conditions `<condition1>` et `<condition2>` est v´erifi´ee.

Par exemple, ´ecrivons un algorithme qui demande a` l'utilisateur de saisir deux valeurs, et qui lui dit si le produit de ces deux valeurs est positif ou n´egatif sans en calculer le produit.

```
Algorithme : Signe du produit
variables:
num´eriques : a, b
DEBUT
    Afficher "Saississez deux valeurs numériques" Saisir a, b
    Afficher "Le produit de ", a, " par ", b, " est " si (a < 0 et b < 0) ou (a ~ 0 et b ~ 0) alors
      Afficher "positif ou nul"
    sinon
        Afficher "négatif"
    fin FIN
```

L'instruction `Afficher "positif ou nul"` sera ex´ecut´ee si au moins une des deux conditions suivantes est v´erifi´ee :

– a < 0 et b < 0
– a ~ 0 et b ~ 0

1.2.2 Suivant cas

Lorsque que l'on souhaite conditionner l'ex´ecution de plusieurs ensembles d'instructions par la valeur que prend une variable, plutôt que de faire des imbrications de `si` a` outrance, on pr´ef´erera la forme suivante :

```
suivant < variable> faire
    cas < valeur1 > < instructions1 >
    cas < valeur2 > < instructions2 >
    ...
    cas < valeurn >
    autres cas < instructions>
finSelon
```

Selon la valeur que prend la variable `<variable>`, le bloc d'instructions a` ex´ecuter est s´electionn´e. Par exemple, si la valeur de `<variable>` est `<valeur 1>`, alors le bloc `<instructions 1>` est ex´ecut´e. Le bloc `<autres cas>` est ex´ecut´e si la valeur de `<variable>` ne correspond a` aucune des valeurs ´enum´er´ees.

Exemple

Ecrivons un algorithme demandant a` l'utilisateur le jour de la semaine. Affichons ensuite le jour ´

correspondant au lendemain.

```
Algorithme : Lendemain
variables: numériques : erreur
alphanumériques : jour, lendemain
DEBUT
    Afficher "Saisissez un jour de la semaine"
    Saisir jour erreur ←- 0
    suivant var faire
        cas "lundi" : lendemain ←- "mardi"
        cas "mardi" : lendemain ←- "mercredi"
                                cas "mercredi" : lendemain ←- "jeudi"
                                cas "jeudi" : lendemain ←- "vendredi"
        cas "vendredi" : lendemain ←- "samedi" cas "samedi" : lendemain ←-
        "dimanche" cas "dimanche" : lendemain ←- "lundi"
        autres cas erreur ←- 1 finSelon
    si erreur = 1 alors
        Afficher "Erreur de saisie"
    sinon
                                        Afficher "Le lendemain du ", jour, " est ", lendemain, "."
    fin FIN
```

Vous remarquez que si l'on avait voulu écrire le même algorithme avec des `Si`, des imbrications nombreuses et peu élégantes auraient éténécessaires.

1.3 Boucles

Une boucle permet d'ex´ecuter plusieurs fois de suite une même s´equence d'instructions. Cette ensemble d'instructions s'appelle le corps de la boucle. Chaque ex´ecution du corps d'une boucle s'appelle une itération, ou encore un passage dans la boucle. Il existe trois types de boucle:

– Tant que

– R´ep´eter ... jusqu'à– Pour

Chacune de ces boucles a ses avantages et ses inconv´enients. Nous les passerons en revue ult´erieurement.

1.3.1 Tant que

La syntaxe d'une boucle `Tant que` est la suivante.

```
                    tant que < condition>
                        < instructions>
 fin tant que
```

La condition est ´evalu´ee avant chaque passage dans la boucle, a` chaque fois qu'elle est v´erifi´ee, on ex´ecute les instructions de la boucle. Un fois que la condition n'est plus v´erifi´ee, l'ex´ecution se poursuit après le `fin tant que`. Affichons par exemple tous les nombres de 1 a` 5 dans l'ordre croissant,

```
Algorithme : 1 a` 5 Tant que
variables: num´erique :i D E B U T
   i ←− 1
   tant que i 5
      Afficher i
      i ←− i + 1 fin tant que
FIN
```

Cet algorithme initialise i a` 1 et tant que la valeur de i n'excède pas 5, cette valeur est affich´ee puis incr´ement´ee. Les instructions se trouvant dans le corps de la boucle sont donc ex´ecut´ees 5 fois de suite. La variable i s'appelle un compteur, on gère la boucle par incr´ementations successives de i et on sort de la boucle une fois que i a atteint une certaine valeur. L'initialisation du compteur est très importante! Si vous n'initialisez pas i explicitement, alors cette variable contiendra n'importe quelle valeur et votre algorithme ne se comportera pas du tout comme pr´evu.

1.3.2 Répéter ... jusqu'à

```
  Répéter
      < instructions>
  jusqu'à< condition>;
```

La fonctionnement est analogue a` celui de la boucle `tant que` a` quelques d´etails près :

– la condition est ´evalu´ee après chaque passage dans la boucle.

– On ex´ecute le corps de la boucle jusqu'àce que la condition soit v´erifi´ee, donc tant que la condition est fausse.

Une boucle `Répéter ... jusqu'`a est donc ex´ecut´ee donc au moins une fois. Reprenons l'exemple pr´ec´edent avec une boucle `Répéter ... jusqu'`a :

```
Algorithme : 1 a` 5 Répéter . . . jusqu'àvariables:

numérique : i
D E B U T
   i ←− 1
   répéter
                                                                    Afficher i
                                                                 i ←− i + 1
                                                              jusqu'ài > 5
FIN
```

De la même façon que pour la boucle `Tant que`, le compteur est initialiséavant le premier passage dans la boucle. Par contre, la condition de sortie de la boucle n'est pas la même, on ne sort de la boucle qu'un fois que la valeur 5 a étéaffichée. Or, i est incrémentée après l'affichage, par conséquent i aura la valeur 6 a` la fin de l'itération pendant laquelle la valeur 5 aura étéaffichée. C'est pour cela qu'on ne sort de la boucle qu'une fois que i a dépasséstrictement la valeur 5. Un des usages les plus courant de la boucle `Répéter ... jusqu'`a est le contrôle de saisie :

```
Répéter
   Afficher "Saisir un nombre strictement positif" Saisir i
   si i ≤ 0 alors
                                          Afficher "J'ai dit STRICTEMENT POSITIF !"
   fin
jusqu'ài > 0
```

1.3.3 Pour

```
   pour < variable> allant de < premierevaleur > a` < dernierevaleur > [par pas de
   < pas >]
      < instructions>
   fin pour
```

La boucle Pour fait varier la valeur du compteur `<variable>` entre `<premiere valeur` et `<derniere valeur>`. Le `<pas>` est optionnel et permet de préciser la variation du compteur entre chaque itération, le pas par défaut est 1 et correspond donc a` une incrémentation. Toute boucle `pour` peut etre réécrite avec une boucle `tant que`. On réécrit de la façon suivante :

```
   < variable >←−< premierevaleur >
   tant que < variable ><>< dernierevaleur > + < pas >
      < instructions>
      < variable >←−< variable> + < pas >
   fin tant que
```

La boucle `pour` initialise le compteur `<variable>` a` la `<premiere valeur>`, et tant que la dernière valeur n'a pas étéatteinte, les `<instructions>` sont exécutées et le compteur incrémentéde `<pas>` si le pas est positif, de `−<pas>` si le pas est négatif.

```
Algorithme : 1 a` 5 Pour
variables:
numérique : i
DEBUT
    pour i allant de 1 a` 5
        Afficher i
    fin pour
FIN
```

Observez les similitudes entre cet algorithme et la version utilisant la boucle `tant que`. Notez bien que l'on utilise une boucle `pour` quand on sait en rentrant dans la boucle combien d'itérations devront être faites. Par exemple, n'utilisez pas une boucle `pour` pour contrôler une saisie!

1.4 Tableaux

Consid´erons un algorithme dont l'ex´ecution donnerait :

```
Saisissez 10 valeurs :
4
90
5
-2
0
6 8
1 -
7
39
Saisissez une valeur :
-7
-7 est la 9-ième valeur saisie.
```

Comment r´ediger un tel algorithme sans utiliser dix variables pour stocker les 10 valeurs ?

1.4.1 D´efinition

Une tableau est un regroupement de variables de même type, il est identifi´e par un nom. Chacune des variables du tableau est num´erot´ee, ce num´ero s'appelle un indice. Chaque variable du tableau est donc caract´eris´ee par le nom du tableau et son indice.

Si par exemple, T est un tableau de 10 variables, alors chacune d'elles sera num´erot´ee et il sera possible de la retrouver en utilisant simultan´ement le nom du tableau et l'indice de la variable. Les diff´erentes variables de T porteront des num´eros de 1 a` 10, et nous appellerons chacune de ces variables un ´el´ement de T.

Une variable n'´etant pas un tableau est appel´ee variable scalaire, un tableau par opposition a` une variable scalaire est une variable non scalaire.

1.4.2 D´eclaration

Comme les variables d'un tableau doivent être de même type, il convient de pr´eciser ce type au moment de la d´eclaration du tableau. De même, on pr´ecise lors de la d´eclaration du tableau le nombre de variables qu'il contient. La syntaxe est :

```
< type >:< nom> [< taille >]
```

Par exemple,

```
numerique : T[4]
```

d´eclare un tableau `T` contenant 4 variables de type num´erique.

1.4.3 Accès aux ´el´ements

Les ´el´ements d'un tableau a` n ´el´ements sont indic´es de 1 a` n. On note `T[i]` l'´el´ement d'indice i du tableau `T`. Les quatre ´el´ements du tableau de l'exemple ci-avant sont donc not´es `T[1]`, `T[2]`, `T[3]` et `T[4]`.

1.4.4 Exemple

Nous pouvons maintenant r´ediger l'algorithme dont le comportement est d´ecrit au d´ebut du cours. Il est n´ecessaire de stocker 10 valeurs de type entier, nous allons donc d´eclarer un tableau `E` de la sorte :

```
numerique : E[10]
```

La d´eclaration ci-dessus est celle d'un tableau de 10 ´el´ements de type num´erique appel´e `E`. Il convient ensuite d'effectuer les saisies des 10 valeurs. On peut par exemple proc´eder de la sorte :

```
Afficher "Saisissez dix valeurs : "
Saisir E[1]
Saisir E[2]
Saisir E[3]
Saisir E[4]
Saisir E[5]
Saisir E[6]
Saisir E[7]
Saisir E[8]
Saisir E[9]
Saisir E[10]
```

La redondance des instructions de saisie de cet extrait sont d'une laideur r´edhibitoire. Nous proc´ederons plus ´el´egamment en faisant une boucle:

```
pour i allant de 1 a` 10
    Saisir E[i]
fin pour
```

Ce type de boucle s'appelle un parcours de tableau. En règle g´en´erale on utilise des boucles pour manier les tableaux, celles-ci permettent d'effectuer un traitement sur chaque ´el´ement d'un tableau. Ensuite, il faut saisir une valeur a` rechercher dans le tableau :

```
Afficher "Saisissez une valeur : "
Saisir t
```

Nous allons maintenant rechercher la valeur `t` dans le tableau `E`. Consid´erons pour ce faire la boucle suivante :

```
i ←- 1
tant que E[i] <> t
    i ←- i + 1
fin tant que
```

Cette boucle parcourt le tableau jusqu'àtrouver un ´el´ement de `E` qui ait la même valeur que `t`. Le problème qui pourrait se poser est que si `t` ne se trouve pas dans le tableau `E`, alors la boucle pourrait ne pas s'arrêter. Si `i` prend des valeurs strictement plus grandes que 10, alors il se produira ce que l'on appelle un débordement d'indice. Vous devez toujours veiller a` ce qu'il ne se produise pas de d´ebordement d'indice ! Nous allons donc faire en sorte que la boucle s'arrête si `i` prend des valeurs strictement sup´erieures a` 10.

```
i ←- 1
tant que i <= 10 et E[i] <> t
    i ←- i + 1
fin tant que
```

Il existe donc deux façons de sortir de la boucle :

– En cas de d´ebordement d'indice, la condition `i <= 10` ne sera pas v´erifi´ee. Une fois sorti de la boucle, `i` aura la valeur 11.

– Dans le cas où `t` se trouve dans le tableau à l'indice `i`, alors la condition `E[i] <> t` ne sera pas vérifiée et on sortira de la boucle. Un fois sorti de la boucle, `i` aura comme valeur l'indice de l'élément de `E` qui est égal à `t`, donc une valeur comprise entre 1 et 10.

On identifie donc la façon dont on est sorti de la boucle en testant la valeur de `i` :

```
si i = 11 alors
    Afficher t, " ne fait pas partie des valeurs saisies." sinon
    Afficher t, " est la ", i, "-ème valeur saisie."
Fin
```

Si `(i = 11)`, alors nous sommes sorti de la boucle parce que l'élément saisi par l'utilisateur ne trouve pas dans le tableau. Dans le cas contraire, `t` est la `i`-ème valeur saisie par l'utilisateur. Récapitulons :

```
Algorithme : Exemple tableau
variables:
numériques : E[10], i, t
DEBUT
    Afficher "Saisissez dix valeurs : "
    pour i allant de 1 a` 10
        Saisir E[i]
    fin pour
    Afficher "Saisissez une valeur : "
    Saisir t
    i ←- 1
    tant que i <= 10 et E[i] <> t
        i ←- i + 1
    fin tant que
    si i = 11 alors
        Afficher t, " ne fait pas partie des valeurs saisies." sinon
        Afficher t, " est la ", i, "-ème valeur saisie."
    fin FIN
```

1.5 Sous-programmes

Il est souvent necessaire, pour eviter d'ecrire plusieurs fois le même code, de preparer des blocs de code prêts a` l'emploi. Une procedure est un ensemble d'instructions. Chaque procedure a un nom, on se sert de ce nom pour executer les instructions contenues dans la procedure.

1.5.1 Principe

```
procedure nomprocedure()
variables :
declaration de variables
DEBUT
  corps de la procedure
FIN
```

Par exemple,

```
procedure afficheBonjour()
DEBUT
  Afficher "bonjour !"
FIN
```

On invoque une procedure en utilisant son nom. Par exemple,

```
afficheBonjour()
```

execute la procedure `afficheBonjour`, on dit aussi qu'on appelle la procedure `afficheBonjour`.

1.5.2 Passage de paramètres

Les variables du programme appelant ne sont pas lisibles depuis un sous-programme. Mais si une procedure, pour s'executer, a` besoin de la valeur d'une variable definie dans le programme appelant, on utilise pour la lui communiquer un mecanisme dit "passage de paramètres".

```
procedure afficheVariable(donnee numerique : monEntier)
DEBUT
  Afficher "La valeur de l'entier passeen param`etre est ", monEntier
FIN
```

La valeur contenue dans la variable monEntier est communiquee au sous-programme lors de l'appel de la procedure. Par exemple,

```
afficheEntier(3)
```

va placer la valeur 3 dans la variable `monEntier`. Il et aussi possible de passer en paramètre une valeur contenue dans une variable. Par exemple, celle d'une variable A :

```
afficheEntier(A)
```

La procedure suivante permute les valeurs des deux variables passees en paramètres :

```
procedure echange(donnees numeriques : x, y)
Variable :
numerique : temp
DEBUT
  temp <- x
x <- y
y <- temp
FIN
```

On peut appeler cette procédure de la fa¸con suivante :

```
echange(A, B)
```

Cependant, l'exécution de ce sous-programme laisse les valeurs de A et B inchangées, car x et y sont des copies de A et B. Les valeurs des copies sont echangées, mais les originaux ne sont pas affectés par cette modification. Si l'on veut passer en paramètres non pas les valeurs des variables, mais les variables elles-mêmes, il faut quelque peu modifier la syntaxe de la procédure :

```
procedure echange(donnees/resultats numeriques : x, y)
Variable :
numerique : temp
DEBUT
  temp <- x
x <- y
y <- temp
FIN
```

1.5.3 Fonctions

Une fonction est un sous-programme quelque peu particulier, dans le sens mathématique du terme, il sert a` calculer une valeur (ie. image). Définissons une fonction simple :

```
fonction successeur(donnee numerique : x)
DEBUT
retourner (x + 1);
FIN
```

Cette fonction "fabrique" le x + 1. On dit que cette fonction renvoie (ou retourne) x + 1. On peut appeler cette fonction ainsi :

```
Afficher x, " + 1 = ", successeur(x)
```

Une fois la fonction appelée, on remplace successeur(x) par le résultat retournépar la fonction.

1.6 Chaˆınes de caractères

1.6.1 Syntaxe

Ca arrive...

Chapitre 2

Exercices

2.1 Introduction

2.1.1 Affectations

Exercice 1 - Jeu d'essai

Après les affectations suivantes :

```
A <- 1
B <- A + 1
A <- B + 2
B <- A + 2
A <- B + 3
B <- A + 3
```

Quelles sont les valeurs de A et de B?

Exercice 2 - Permutation des valeurs de 2 variables

Quelle série d'instructions échange les valeurs des deux variables A et B déjàinitialisées ?

2.1.2 Saisie, affichage, affectations

Exercice 3 - Nom et ˆage

Saisir le nom et l'âge de l'utilisateur et afficher "Bonjour ..., tu as ... ans." en remplaçant les ... par respectivement le nom et l'âge.

Exercice 4 - Permutation de 2 variables saisies

Saisir deux variables et les permuter avant de les afficher.

Exercice 5 - Moyenne de 3 valeurs

Saisir 3 valeurs, afficher leur moyenne.

Exercice 6 - Aire du rectangle

Demander a` l'utilisateur de saisir les longueurs et largeurs d'un rectangle, afficher sa surface.

Exercice 7 - Permutation de 4 valeurs

Ecrire un algorithme demandant a` l'utilisateur de saisir 4 valeurs A, B, C, D et qui permute les variables de la façon suivante :

noms des variables	A	B	C	D
valeurs avant la permutation	1	2	3	4
valeurs après la permutation	3	4	1	2

Exercice 8 - Permutation de 5 valeurs

On considère la permutation qui modifie cinq valeurs de la façon suivante :

noms des variables	A	B	C	D	E
valeurs avant la permutation	1	2	3	4	5
valeurs après la permutation	4	3	5	1	2

Ecrire un algorithme demandant a` l'utilisateur de saisir 5 valeurs que vous placerez dans des variables appelées A, B, C, D et E. Vous les permuterez ensuite de la façon décrite ci-dessus.

Exercice 9 - Pièces de monnaie

Nous disposons d'un nombre illimitéde pièces de 0.5, 0.2, 0.1, 0.05, 0.02 et 0.01 euros. Nous souhaitons, étant donnéune somme S, savoir avec quelles pièces la payer de sorte que le nombre de pièces utilisée soit minimal. Par exemple, la somme de 0.96 euros se paie avec une pièce de 0.5 euros, deux pièces de 0.2 euros, une pièce de 0.05 euros et une pièce de 0.01 euros.

1. Le fait que la solution donnée pour l'exemple est minimal est justifiépar une idée plutôt intuitive. Expliquez ce principe sans excès de formalisme.
2. Ecrire un algorithme demandant a` l'utilisateur de saisir une valeur positive ou nulle. Ensuite, affichez le détail des pièces a` utiliser pour constituer la somme saisie avec un nombre minimal de pièces.

Vous utiliserez l'instruction partieEntiere pour tronquer les quotients des divisions. Par exemple,

```
A ←— partieEntiere(3.2)
```

place dans A la valeur 3.

2.2 Traitements conditionnels

2.2.1 Exercices de compr´ehension

Que font les suites d'affectations suivantes ?

Exercice 1

```
A ←− 1;
B ←− 2;
si A ≥ B alors
    A ←− B; sinon
    B ←− A;
Fin
```

2.2.2 Conditions simples

Exercice 2 - Majorité Saisir l'âge de l'utilisateur et lui dire s'il est majeur.

Exercice 3 - Valeur absolue

Saisir une valeur, afficher sa valeur absolue. On rappelle que la valeur absolue de x est la distance entre x et 0.

Exercice 4 - Admissions

Saisir une note, afficher "ajourné" si la note est inférieure a` 8, oral entre 8 et 10, admis dessus de 10.

Exercice 5 - Assurances

Une compagnie d'assurance effectue des remboursements sur lesquels est ponctionnée une franchise correspondant a` 10% du montant a` rembourser. Cependant, cette franchise ne doit pas excéder 4000 euros. Demander a` l'utilisateur de saisir le montant des dommages, afficher ensuite le montant qui sera remboursé ainsi que la franchise.

Exercice 6 - Assurances

Une compagnie d'assurance effectue des remboursements en laissant une somme, appelée franchise, a la charge du client. La franchise représente 10% du montant des dommages sans toutefois pouvoir ˆetre inférieure a` 15 euros ou supérieure a` 500 euros. Ecrire un algorithme demandant a` l'utilisateur ˆde saisir le montant des dommages et lui affichant le montant remboursé ainsi que le montant de la franchise.

Exercice 7 - Valeurs distinctes parmi 2

Afficher sur deux valeurs saisies le nombre de valeurs disctinctes.

Exercice 8 - Plus petite valeur parmi 3

Afficher sur trois valeurs saisies la plus petite.

Exercice 9 - Recherche de doublons

Ecrire un algorithme qui demande a` l'utilisateur de saisir trois valeurs et qui lui dit s'il s'y trouve un doublon.

Exercice 10 - Tri de 4 valeurs

Ecrire un algorithme demandant a` l'utilisateur de saisir 4 valeurs et qui les affiche dans l'ordre croissant.

Exercice 11 - Impôts

Les impôts en france sont calculés avec un système de tranches. Les montants et taux utilisés dans cet exercice ne sont pas les mêmes que ceux utilisés par le Trésor Public, mais le principe est le même. Soit M le revenu imposable, pour calculer le montant de l'impôt, on commence par découper M en tranches de 2000 euros. Seule la dernière tranche peut valoir moins de 2000 euros. Par exemple, la somme 5637 euros est découpéen trois tranches de 2000, 2000 et 1637 euros. La première tranche n'est pas imposée, la deuxième est imposée au taux de 10%, la troisième est imposée a` 15%, la quatrième est imposée a` 20%. A partir de la cinquième tranche, le taux d'imposition est 25%. Par exemple, si le revenu imposable est 5637 euros, la première tranche n'est pas imposée, le montant de l'impôt sur la deuxième tranche est 2000*0.1 = 200 euros et le montant de l'impôt sur la troisième tranche est 1637* 0.15 = 245.55 euros. Le montant de l'impôt sera donc 200 + 245.55 = 445.55 euros. Ecrire un algorithme demandant a` l'utilisateur de saisir son revenu imposable et calculant le montant des impôts qu'il devra payer.

2.2.3 Conditions imbriquées

Exercice 12 - Signe du produit

Saisir deux nombres et afficher le signe de leur produit sans les multiplier.

Exercice 13 - Valeurs distinctes parmi 3

Afficher sur trois valeurs saisies le nombre de valeurs disctinctes.

Exercice 14 - ax + b = 0

Saisir les coefficients a et b et afficher la solution de l'équation ax + b = 0.

Exercice 15 - $ax^2 + bx + c = 0$

Saisir les coefficients a, b et c, afficher la solution de l'équation $ax^2 + bx + c = 0$.

Exercice 16 - Op´erations sur les heures

Ecrire un algorithme qui demande a` l'utilisateur de saisir une heure de début (heures + minutes) et une heure de fin (heures + minutes aussi). Cet algorithme doit ensuite calculer en heures + minutes le temps écouléentre l'heure de début et l'heure de fin. Si l'utilisateur saisit 10h30 et 12h15, l'algorithme doit lui afficher que le temps écouléentre l'heure de début et celle de fin est 1h45. On suppose que les deux heures se trouvent dans la même journée, si celle de début se trouve après celle de fin, un message d'erreur doit s'afficher. Lors la saisie des heures, séparez les heures des minutes en demandant a` l'utilisateur de saisir :

heures de début
– minutes de début
– heures de fin
minutes de fin

Exercice 17 - Le jour d'après

Ecrire un algorithme de saisir une date (jour, mois, ann´ee), et affichez la date du lendemain. Saisissez les trois donn´ees s´epar´ement (comme dans l'exercice pr´ec´edent). Prenez garde aux nombre de jours que comporte chaque mois, et au fait que le mois de f´evrier comporte 29 jours les ann´ees bissextiles. Une ann´ee est bissextile si elle est divisible par 4 mais pas par 100 (`http://fr.wikipedia.org/wiki/Ann\%C3\%A9e_bissextile)`. Vous aurez besoin de calculer le reste de la division de p par q, pour ce faire, utiliser l'instruction `reste(p, q)`. Par exemple, après l'affectation :

```
A ←- reste(22, 7) ;
```

La variable A contient la valeur 1.

2.2.4 L'échiquier

On indice les cases d'un ´echiquer avec deux indices i et j variant tous deux de 1 a` 8. La case (i,j) est sur la ligne i et la colonne j. Par convention, la case (1, 1) est noire.

Exercice 18 - Couleurs

Ecrire un programme demandant a` l'utilisateur de saisir les deux coordonn´ees i et j d'une case, et lui disant s'il s'agit d'une case blanche ou noire.

Exercice 19 - Cavaliers

Ecrire un programme demandant a` l'utilisateur de saisir les coordonn´ees (i, j) d'une première case et les coordonn´ees ($i^{'}$,$j^{'}$) d'une deuxième case. Dites-lui ensuite s'il est possible de d´eplacer un cavalier de (i,j) a` ($i^{'}$,$j^{'}$).

Exercice 20 - Autres pièces

Donner des conditions sur (i, j) et ($i^{'}$, $j^{'}$) permettant de tester la validit´e d'un mouvement de tour, de fou, de dame ou de roi.

2.2.5 Suivant Cas

Exercice 21 - Calculatrice

Ecrire un algorithme demandant a` l'utilisateur de saisir deux valeurs num´eriques a et b, un op´erateur op (v´erifier qu'il s'agit de l'une des valeurs suivantes : +, -, *, /) de type alphanum´erique, et qui affiche le r´esultat de l'op´eration a op b.

2.3 Boucles

2.3.1 Utilisation de toutes les boucles

Les exercices suivants seront rédigés avec les trois types de boucle : tant que, répéter jusqu'àet pour.

Exercice 1 - compte a` rebours

Ecrire un algorithme demandant a` l'utilisateur de saisir une valeur numérique positive n et affichant toutes les valeurs n, n − 1,. .. ,2,1,0.

Exercice 2 - factorielle

Ecrire un algorithme calculant la factorielle d'un nombre saisi par l'utilisateur.

Exercice 3

Repérer, dans les exercices de la section précédente, les types de boucles les plus adaptées au problème.

2.3.2 Choix de la boucle la plus appropriée

Pour les exercices suivants, vous choisirez la boucle la plus simple et la plus lisible.

Exercice 4

Ecrire un algorithme demandant a` l'utilisateur de saisir la valeur d'une variable n et qui affiche la table de multiplication de n.

Exercice 5 - puissance

Ecrire un algorithme demandant a` l'utilisateur de saisir deux valeurs numériques b et n (vérifier que n est positif) et affichant la valeur b^{Th}.

Exercice 6

Ecrire un algorithme demandant a` l'utilisateur de saisir la valeur d'une variable n et qui affiche la valeur 1 + 2 + ... + (n − 1) + n.

Exercice 7 - nombres premiers

Ecrire un algorithme demandant a` l'utilisateur de saisir un nombre au clavier et lui disant si le nombre saisi est premier.

Exercice 8 - somme des inverses

Ecrivez un algorithme saisissant un nombre n et calculant la somme suivante :

$$1 - \frac{1}{2} + \frac{1}{3} + \ldots + \frac{(-1)^{Th+1}}{n}$$

Vous remarquez qu'il s'agit des inverses des n premiers nombres entiers. Si le dénominateur d'un terme est impair, alors vous l'aditionnez aux autres, sinon vous le soutrairez aux autres. Exercice 9 - n_{Th}

Ecrire un algorithme demandant la saisie d'un nombre n et calculant n^{Th}. Par exemple, si l'utilisateur saisit 3, l'algorithme lui affiche $3^3 = 3 \times 3 \times 3 = 27$.

Exercice 10 - racine carrée par dichotomie

Ecrire un algorithme demandant a` l'utilisateur de saisir deux valeurs numériques x et p et affichant $\sqrt{x}$ avec une précision p. On utilisera une méthode par dichotomie : a` la k-ème itération, on cherche x dans l'intervalle [min, sup], on calcule le milieu m de cet intervalle (`a vous de trouver comment la calculer). Si cet intervalle est suffisament petit (`a vous de trouver quel critère utiliser), afficher m. Sinon, vérifiez si $\sqrt{x}$ se trouve dans [inf, m] ou dans [m, sup], et modifiez les variables inf et sup en conséquence. Par exemple, calculons la racine carrée de 10 avec une précision 0.5,

- Commençons par la chercher dans [0, 10], on a m = 5, comme $5^2 > 10$, alors $5 > \sqrt{10}$, donc $\sqrt{10}$ se trouve dans l'intervalle [0, 5].
- On recommence, m = 2.5, comme $5\ 2 = 25\ 4 < 10$, alors $5\ 2 < \sqrt{10}$, on poursuit la recherche dans [5
- On a m = 3.75, comme $3.75^2 > 10$, alors $3.75 > \sqrt{10}$ et $\sqrt{10}$ [2.5, 3.75]
- On a m = 3.125, comme $3.125^2 < 10$, alors $3.75 < \sqrt{10}$ et $\sqrt{10} \in$ [3.125, 3.75]
- Comme l'étendue de l'intervalle [3.125, 3.75] est inférieure 2×0.5, alors m = 3.4375 est une approximattion a` 0.5 près de $\sqrt{10}$.

2.4 Tableaux

2.4.1 Prise en main

Exercice 1 - Initialisation et affichage

Ecrire un algorithme les valeurs 1, 2,3,.. . , 7 dans un tableau T a` 7 ´el´ement, puis affichant les el´ements de T en partant de la fin.
´

Exercice 2 - Contrôle de saisie

Ecrire un algorithme plaçant 20 valeurs positives saisies par l'utilisateur dans un tableau a` 20 el´ements. Vous refuserez toutes les valeurs strictement n´egatives.
´

Exercice 3 - Choix des valeurs supérieures a` t
´

Ecrire un algorithme demandant a` l'utilisateur de saisir dix valeurs num´eriques puis de saisir une valeur t. Il affichera ensuite le nombre de valeurs strictement sup´erieures a` t. Par exemple, si l'utilisateur saisit 4, 19, 3, −2, 8, 0, 2, 10,34, 7 puis 3, alors le nombre de valeurs strictement sup´erieures a 3 parmi les 10 premi`eres saisies est 6 (4, 19, 8, 10, 34 et 7).
`

2.5 Sous-programmes

2.5.1 Fonctions

Exercice 1 - Analyse combinatoire

1. On note n! le nombre 1.2.3.4... (n − 1).n. Par exemple, 5! = 1.2.3.4.5 = 120 et on a par convention 0! = 1. Ecrivez une fonction `factorielle(numerique : n) : numerique` retournant la factorielle du nombre n passéen paramètre.
2. Ecrivez une fonction `puissance(numeriques : b, n) : numerique` retournant b^n, oübest un entier non nul et n un entier positif. N'oubliez pas que $b^0 = 1$.
3. On note $A_{p\ n}$ le nombre (n − p + 1)(n − p + 2)... (n − 1)n. Par exemple, $A_{4\ 6}$ = 3.4.5.6 = 360. Ecrivez une fonction `arrrangements(numeriques : p, n) : numerique` retournant $A_{p\ n}$ si p ≤ n et −1 sinon.
4. On note $C_{p\ n}$ le nombre $\frac{(n-p+1)(n-p+2)\ldots(n-1)n}{1.2\ldots(p-1).p}$. Par exemple, $C_{4\ 6} = \frac{3.4.5.6}{1.2.3.4} = 15$.
   Ecrivez une fonction `combinaisons(numeriques : p, n) : numerique` retournant $C_{p\ n}$ si p ≤ n et −1 sinon.

Exercice 2 - Nombres premiers

Cet exercice a pour but l'écriture d'un algorithme de recherche du i-ème nombre premier. Rappelons qu'un nombre est premier s'il n'est divisible que par 1 et par lui-même. Vous prendrez garde au fait que 1 n'est pas considérécomme un nombre premier, le premier nombre premier est donc 2, le deuxième est 3, le troisième est 5, etc.

1. Ecrivez la fonction `estPremier(numerique : n) : numerique` prenant en paramètre un entier n, cette fonction retourne 1 si n est premier, −1 sinon.
2. Ecrivez une fonction `plusPetitPremier(numerique : m) : numerique` prenant en paramètre un entier m, et retournant le plus petit nombre premier n tel que m ≤ n. Vous utiliserez la fonction `estPremier`
3. Ecrivez une fonction `trouvePremier(numerique : i) : numerique` prenant en paramètre un entier i, et retournant le i-ème nombre premier. Vous utiliserez pour ce faire la fonction `plusPetitPremier`.
4. Ecrivez un algorithme demandant a` l'utilisateur de saisir un nombre j et lui affichant le j-ème nombre premier. Vous utiliserez la fonction `trouvePremier`.

2.5.2 Sous-programmes et tableaux

Exercice 3 - Pour commencer

1. Ecrire la fonction `contient(numeriques : T[N], x) : booleen`. `contient(T, x)` retourne vrai si et seulement si le tableau T contient l'élément x.
2. Ecrire la procedure `remplace(numeriques : T[N] e/s, x, y)`. `remplace(T, x, y)` remplace dans T toutes les occurrences de x par des y.

Exercice 4 - indices

1. Ecrire la fonction `valeursDistinctes(numeriques : T[n]) : numerique`. Cette fonction retourne le nombre de valeurs distinctes contenues dans le tableau T, c'est-à-dire le nombre de valeurs contenues dans T une fois tous les doublons supprimés. Par exemple, les valeurs distinctes de [2,4,8, 2, 7,3,8,2, 5,0,8,4, 1] sont {2, 4,8,7,3,5,0, 1}, le nombre de valeurs distinctes est donc 8.

2. Ecrire la proc´edure carres(numerique : T[n] e/s) : numerique. Cette proc´edure place dans T les carr´es dans n premiers nombres entiers. Vous utiliserez le fait que k^2 est la somme des k premiers entiers impairs. Par exemple, $3^2 = 1 + 3 + 5 = 9$, $5^2 = 1 + 3 + 5 + 7 + 9 = 25$, etc.
3. Ecrire la proc´edure cubes(numeriques : T[n] e/s) : numerique. Cette proc´edure place dans T les cubes dans n premiers nombres entiers. Vous utiliserez la proc´edure carres et le fait que $k^3 = (k-1)^3 + 3(k-1)^2 + 3(k-1) + 1$. Par exemple, si k = 3, on a $3^3 = (2)^3 + 3(2)^2 + 3(2) + 1 = 8 + 12 + 6 + 1 = 27$.
4. Ecrire la proc´edure interclasse(numeriques : S[n], T[p], Q[n + p]) cette fonction prend deux tableaux tri´es S et T en paramètre. Elle place dans Q les ´el´ements de S et de T dans l'ordre croissant. Si par exemple, S = [1,4,6,8,23,54] et T = [2,3,7, 12, 17,45,52,77], alors interclasse place [1,2,3,4,6,7,8,12,17,23,45,52,54,77] dans Q.

Exercice 5 - initiation a` l'encapsulation

Nous allons cr´eer un ensemble de sous programmes permettant de manipuler des tableaux tri´es. Ce tableaux sont de taille N, vous prendrez le soin de v´erifier qu'il ne se produit aucun d´ebordement d'indice. Nous voulons pouvoir faire varier le nombre d''´elements plac´es dans le tableau, c'est-à-dire en ajouter et en enlever. Comme les tableaux sont de taille fixe, il convient de placer dans une variable le nombre d''´el´ements significatifs du tableau. On utilisera pour ce faire la première case du tableau, c'est-à-dire celle d'indice 1. Par exemple, le tableau

0	5	3	6	2	4	5	8	3	6	2	7	3	4	0

ne contient aucun ´el´ement, le 0 qui se trouve au d´ebut du tableau nous indique qu'aucun ´el´ement n'est significatif. Les ´el´ements significatifs sont plac´es juste après le premier ´el´ement. Donc, s'il y a 7 ´el´ements significatifs dans le tableau T, on aura T(1) = 7, et ces 7 ´el´ements seront stock´es dans T(2),.. . , T(8). Par exemple, le tableau

3	2	3	6	2	4	5	8	3	6	2	7	3	4	0

contient les 3 ´el´ements 2, 3 et 6. Le 3 se trouvant dans le premier emplacement du tableau nous indique que seuls les 3 premiers ´el´ements sont significatifs, les autres sont ignor´es. Notez que nous ferons en sorte que les ´el´ements significatifs soient dispos´es par ordre croissant.

1. Ecrire la proc´edure affiche(numerique : T[N]). Ce sous programme affiche tous les ´el´ements significatifs du tableau T.
2. Ecrire la fonction taille(numerique : T[N]) : numerique. Ce sous programme retourne le nombre d''´el´ements significatifs se trouvant dans le tableau T. Ne cherchez pas quelquechose de compliqué, il y a une seule instruction dans ce sous-programme!
3. Ecrire la fonction estVide(numerique : T[N]) : booleen. Ce sous programme retourne vrai si et seulement si le tableau T est vide.
4. Ecrire la fonction estPlein(numeriques : T[N]) : booleen. Ce sous programme retourne vrai si et seulement si le tableau T est plein.
5. Ecrire la proc´edure initialise(numeriques : T[N], k). Ce sous programme initialise T de la sorte : | k | 1 | 2 | ... | k | ... | on supposera que k est toujours strictement inf´erieur a` N.
6. Ecrire la fonction indice(numeriques : T[N] e/s, x) : numerique. Ce sous-programme retourne l'indice de l''´el´ement x dans le tableau T, -1 si x ne se trouve pas dans T. On part du principe que le tableau T est trié, vous devez en tenir compte dans votre sous-programme.
7. Ecrire la proc´edure supprimeFin(numeriques : T[N] e/s). Ce sous programme supprime le dernier ´el´ement du tableau en le rendant non significatif. Si le tableau est vide, ce sous-programme ne fait rien.
8. Ecrire la proc´edure ajouteFin(numeriques : T[N] e/s, x). Ce sous-programme ajoute l'´el´ement x a` la fin du tableau T, en v´erifiant que le tableau reste tri´e, et qu'aucun d´ebordement ne se produit. Si l'ajout est impossible, ce sous-programme ne fait rien.

9. Ecrire la proc´edure `decalageGauche(num´erique : T[N] e/s, i, j)`. Ce sous programme d´ecale vers la gauche la tranche T(i),.. . ,T(j). Vous partirez du principe que les valeurs de i et j sont telles qu'aucun d´ebordement d'indice ne peut se produire. Vous ne modifierez pas la valeur de T(1). Si cette proc´edure est appel´ee sur le tableau

3	1	4	6	9	12	16	21	120

avec i = 4 et j = 7, on obtient

3	1	6	9	12	16	16	21	120

Notez bien que ce n'est pas une permutation circulaire.

10. Ecrire la proc´edure `decalageDroite(num´eriques : T[N] e/s, i, j)`. Ce sous programme d´ecale vers la droite la tranche T(i),. .. , T(j). Vous partirez du principe que les valeurs de i et j sont telles qu'aucun d´ebordement d'indice ne peut se produire. Vous ne modifierez pas la valeur de T(1).

11. Ecrire la proc´edure `insere(num´eriques : T[N] e/s, x)`. Ce sous programme insère l'´el´ement x dans le tableau en veillant a` ce que celui-ci soit triéaprès l'insertion. Si le tableau est plein, ou que x est d´ejàdans le tableau, alors ce sous-programme ne fait rien.

12. Ecrire la proc´edure `supprime(num´eriques : T[N] e/s, x)`. Ce sous programme supprime l'´el´ement x du tableau. Si x ne se trouve pas dans T, alors ce sous-programme ne fait rien.

13. Nous souhaitons perfectionner la fonction `indice`. Au lieu de parcourir tous les ´el´ements significatifs du tableau, nous allons les s´eparer en deux tranches et v´erifier dans quelle tranche peut se trouver l'´el´ement x que l'on recherche. Par exemple, si l'on cherche l'´el´ement 4 dans le tableau

9	1	4	6	9	12	14	16	21	120

on commence par l'´el´ement se trouvant au milieu du tableau, a` savoir 12. On s´epare donc les el´ements significatifs en deux tranches, a` savoir

1	4	6	9

et

14	16	21	120

Comme 12 =6 4, il est n´ecessaire de poursuivre la recherche. En comparant 12 a` 4, on d´eduit, parce que le tableau est tri´e, que l'´el´ement 4 ne peut se trouver que dans la tranche

1	4	6	9

des 4 premiers ´el´ements significatifs du tableau. On applique le même proc´ed´e a` cette tranche. On peut couper cette tranche en deux soit de part et d'autre du 4, soit de part et d'autre du 6. Si nous choisissons 6, il reste deux tranches :

1	4

et

9

on poursuit la recherche jusqu'àce qu'il reste un seul ou aucun ´el´ement dans la tranche consid´er´ee. R´e´ecrire la fonction `indice` en utilisant la m´ethode d´ecrite pr´ec´edement.

2.5.3 Département aspirine

Exercice 6 - nombres entiers multiprécision

Les nombres entiers dans les langages de programmation sont limit´es en taille. Par exemple, un `int` en C est cod´e sur 4 octets, il n'est donc pas possible d'y placer des nombres sup´erieurs a` $2^{31} - 1$. Nous allons cr´eer un ensemble de sous-programmes permettant de manipuler des entiers positifs de grande taille. Nous d´ecidons de les repr´esenter par des tableaux, chaque ´el´ement du tableau sera un chiffre. Par exemple, le nombre

153624586273405083236

sera repr´esent´e par le tableau

23	1	5	3	6	2	4	5	8	3	6	2	7	3	4	0	5	0	8	3	2	3	6

Le nombre 23 repr´esente le nombre d'´el´ements significatifs dans le tableau. Les tableaux auront une taille N suppos´ee suffisament importante pour qu'aucun problème de d´epassement de capacit´e se pose. Ecrivez les corps des sous-programmes suivants :

1. proc´edure affiche(num´erique : T[n]). affiche(T) affiche les ´el´ements de T.
2. proc´edure convertit(num´eriques : T[N] e/s, x). convertit(T, n) place dans le tableau T la repr´esentation d´ecimale du nombre x.
3. proc´edure additionne(num´eriques : A[n], B[n], C[n] e/s). additionne(A, B, C) place dans C la repr´esentation d´ecimale de la somme des deux nombres repr´esent´es par A et B. En cas de d´epassement de capacit´e, les nombres les plus a` gauche de C sont ignor´es.
4. proc´edure soustrait(num´eriques : A[n], B[n], C[n] e/s). soustrait(A, B, C) place dans C la repr´esentation d´ecimale de la diff´erence entre les deux nombres repr´esent´es par A et B. On part du principe que cette diff´erence ne peut pas être n´egative.
5. proc´edure mult10n(num´eriques : A[n] e/s, k). mult10n(A, k) place dans A la repr´esentation d´ecimale du produit de A par 10^k.
6. proc´edure multScalaire(num´eriques : A[n] e/s, k) multScalaire(A, k) place dans A la repr´esentation d´ecimale du produit de A par k.
7. proc´edure multiplie(num´eriques : A[n], B[n], C[n] e/s). multiplie(A, k) place dans C la repr´esentation d´ecimale du produit des deux nombres repr´esent´es par A et B. En cas de d´epassement de capacit´e, les nombres les plus a` gauche de C sont ignor´es.

Exercice 7 - récursivité

Un sous-programme r´ecursif est un sous-programme qui s'appelle lui-même. Par exemple, le sous programme

```
Fonction factorielle(n)
   si n = 0 alors
      retourner 1
   sinon
                                                    retourner n x factorielle(n - 1)
   fin FIN
```

calcule la factorielle de n, a` savoir 1 x 2 x ... x (n - 1) x n.

1. Quelle est la valeur retourn´ee par factorielle(5)?
2. Ecrire une fonction r´ecursive calculant b^n, oüb est un nombre non nul et n un entier positif ou nul. N'oubliez pas que $b^0 = 1$

2.6 Tris

Exercice 1 - test

Ecrivez le corps de la fonction `estTri´e(numerique : T[N]) : bool´een`. `estTri´e(T)` retourne vrai si et seulement si le tableau `T` est tri´e par ordre croissant. :

Exercice 2 - tri par s´election

Le tri par s´election est une m´ethode consistant a` rechercher dans un tableau `T` a` n ´el´ements le plus petit ´el´ement du tableau et a` l'´echanger avec le T(1). Puis a` chercher dans T(2),.. . , T(N) le deuxième plus petit ´el´ement et a` l'´echanger avec T(2), etc. Une fois un tri par s´election achev´e, les ´el´ements du tableau doivent être dispos´es par ordre croissant. ´Ecrivez les sous-programmes suivants :

1. `proc´edure ´echange(num´eriques : T[N] e/s, i, j`. Cette proc´edure doit ´echanger les ´el´ements `T(i)` et `T(j)`.
2. `fonction indiceDuMin(num´erique : T[N] e/s, i, j)` retournant l'indice du plus petit el´ement de `T(i), ..., T(j)`, c'est-à-dire de tous les ´el´ements du tableau dont l'indice est compris entre i et j.
3. `proc´edure placeMin(num´eriques : T[N], i, j, k)`. ´echangeant avec `T(k)` le plus petit el´ement de `T` dont l'indice est compris entre i et j.
4. `proc´edure triParS´election(num´eriques : tableau T de N ´el´ements num´eriques)`. `triParS´election(T)` trie le tableau `T`.

2.7 Matrices

Exercice 1 - somme

Ecrire une proc´edure calculant la somme de deux matrices.

Exercice 2 - multiplication

Ecrire une proc´edure calculant la somme de deux matrices.

Exercice 3 - transposition

Ecrire une proc´edure calculant la matrice transpos´ee d'une matrice N x M pass´ee en paramètre.

Exercice 4 - triangle de Pascal

Un triangle de Pascal peut être plac´e dans une matrice, dont seule la partie triangulaire inf´erieure est renseign´ee. La première ligne et la première colonne d'une triangle de Pascal ne contiennent que des 1. Et, si on note P(i,j) la valeur se trouvant dans la i-ème ligne et la j-ème colonne de cette matrice, alors on a

$$m(i,j) = m(i - 1,j - 1) + m(i - 1,j)$$

pour tous i et j sup´erieurs ou ´egaux a` 1. Ecrire une proc´edure initialisant un triangle de Pascal a` n lignes.

Exercice 5 - puissances

Ecrire une proc´edure remplissant une matrice m de la façon suivante :

$$m(i,j) = i^{j-1}$$

Vous utiliserez le fait que

$$i^{j-1} = P(0,j)(i - 1)^0 + P(1,j)(i - 1)^1 + \ldots + P(k,j)(i - 1)^k + \ldots + P(j,j)(i - 1)^j$$

oüP(a, b) est l'´el´ement se trouvant dans la ligne b + 1 et la colonne a + 1 du triangle de Pascal calcul´e dans l'exercice pr´ec´edent.

Exercice 6 - matrices de Toepliz

1. Soit M une matrice a` n lignes et a` p colonnes, on note m(i,j) l'´el´ement de M de trouvant a` la i-ème ligne et a` la j-ème colonne. M est une matrice de Toepliz si pour tout i E {2,... , n} et pour tout j E {2,.. . ,p},

$$m(i,j) = m(i - 1,j) + m(i,j - 1)$$

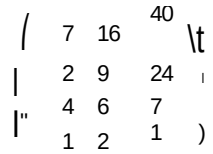
Par exemple :

$$\begin{pmatrix} 1 & 4 & 2 & 1 & 7 \\ 2 & 6 & 8 & 9 & 16 \\ 1 & 7 & 15 & 24 & 40 \\ -2 & 5 & 20 & 44 & 84 \\ 2 & 7 & 27 & 71 & 155 \end{pmatrix}$$

Ecrire la proc´edure `toepliz(num´erique : T[m, n] e/s)` prenant en paramètre une matrice dont la première ligne et la première colonne sont initialis´ees. Cette proc´edure initialise tous les autres ´el´ements de la matrice de sorte que T soit une matrice de Toepliz.

2. on appelle rotation de matrice l'opération qui transforme la matrice

en



Ecrire la procédure `rotation(num´erique : T[m, n], Q[n, m] e/s)` affectant a` la matrice `Q` le résultat de la rotation de la matrice `T`.

3. Ecrire la procédure `rotation(num´erique : T[n, n] e/s)`, prenant en paramètre une matrice carrée `T`, et modifiant cette matrice de sorte qu'elle contienne le résultat de sa rotation.

Chapitre 3

Quelques corrigés

3.1 Boucles

3.1.1 C + /C-

```
Algorithme : C + /C—
variables:
numériques : inconnu, essai
DEBUT
    inconnu ←— random
    répéter
        Afficher "Saisissez une valeur"
        Saisir essai
        si inconnu < essai alors
            Afficher C—
        fin
        si inconnu > essai alors
            Afficher C+
        fin
    jusqu'àinconnu = essai
    Afficher "Gagné!"
FIN
```

3.1.2 Somme des inverses

```
Algorithme : Somme des inverses
variables:
numériques : nbTermes, somme, indice, signe
DEBUT
    Afficher "Saisissez une valeur"
    Saisir nbTermes
    somme ←— 0
    signe ←— 1
    pour indice allant de 1 a~ nbTermes
        somme ←— somme + signe *
        (1/i) signe ←— —signe
    fin pour
    Afficher "S = ", somme
FIN
```

3.1.3 n

```
Algorithme : nn
variables:
numériques : n, i, res
DEBUT
    Afficher "Saisissez une valeur"
    Saisir n
    res ←-
    1
                                    pouri allant de 1 a~ n
                                       res ←- res x n
    fin pour
    Afficher "n'n = ", res
```

3.2 Tableaux

3.2.1 Choix des valeurs supérieures a` t

```
Algorithme : supérieures a` t
variables:
numériques : v[10], t, i, nb
DEBUT
    pour i allant de 1 a~ 10
        Afficher "Saisissez une valeur" Saisir v[i]
    fin pour
    Afficher "Saisissez t"
    Saisir t
    nb <-- 0
    pour i allant de 1 a~ 10
        si v[i] > t alors
            nb <-- nb + 1
        fin
    fin pour
    Afficher "Vous avez saisi ", nb, "valeurs supérieures a~ ", t, "." FIN
```