Algorithmique

Groupe KabritBwa

13 février 2018

"Il ne s'agit ni de rire, ni de pleurer mais de comprendre"
Spinoza

Cette brochure est le fruit des travaux des membres "Kabrit Bwa" du Groupe IREM section Martinique :
Daniel ALPHONSE - Jean-François CULUS - Christian CYRILLE - Tony FRANCIL - Ghislaine LOMBARD - Patrick LOMBARD - Aliette MAINGE - Dominique NAUDIN - Patrick ZOCLY

Elle est spécialement dédiée à tous nos collègues et amis pionniers qui ont consacré des années à l'initiation à la programmation des élèves de l'option informatique des lycées dans l'ancienne Académie Antilles-Guyane de 1985 à 1997

1. GUYANE :
   Daniel SENECHAL, Alain LLAMAS,...
2. GUADELOUPE :
   Mme Bertille ADELAIDE-MERLANDE (Responsable Universitaire à l'UAG de l'Option Informatique Antilles Guyane) ; Daniel BELLIER ; MARCHISIO ; Gustave PIERRE-JEAN, Claude DEJEAN, ...
3. MARTINIQUE :
   Daniel ALPHONSE ; Gabrielle BARBE ; Victor BAUDIN ; Marie-Josée CIRBA ; Lucette CORANSON-BEAUDU ; Christian CYRILLE (deuxième coordonnateur option informatique Antilles Guyane) ; Viviane DELBLOND ; Victor DEVOUE ; Patrick JEAN-BAPTISTE ; Joseph JEREMIE ; Françoise LABRIDY ; Michel LIBEROS ; Daniel MONTLOUIS-CALIXTE ; Dominique METRIE ; Ricardo MERIDA ; Christian PISELLI ; Jean-Claude PORLON ; Louis RICHER ; Michel ROSE-CLAIRE SANON ; Lucien Philippe SABINE ; Frantz ZOZOR-TELGA ...

Un remerciement particulier en particulier :
Georges STAMOND, professeur émérite Université Paris 5, qui a formé avec les professeurs BERGER et DURAND la première génération de professeurs antillo-guyanais

Jacques ARSAC (1929 - 2014) professeur émérite Université Pierre et Marie Curie, Membre de l'Académie des Sciences, Responsable National de l'Option Informatique

Une dédicace spéciale à ceux qui nous ont déjà quitté et qui ont tant contribué au développement de l'informatique dans L'Education Nationale :

- Bernard ALLAUD (1947 - 1987)(Sciences Physiques, premier coordonnateur option informatique Antilles Guyane - Lycée Schoelcher)
- Gustave MARIE-JEANNE (1939 - 2010) (IPR Sciences Physiques, Fondateur de l'option informatique des lycées aux Antilles et à la Guyane - Proviseur Honoraire Lycée Schoelcher)
- Philippe Lucien SABINE (- 2011) (Sciences Physiques - Option informatique - Lycée Schoelcher)
- Emile SERALINE(1924 - 2015) (Chef du Laboratoire de Sciences Physiques - Lycée Schoelcher)

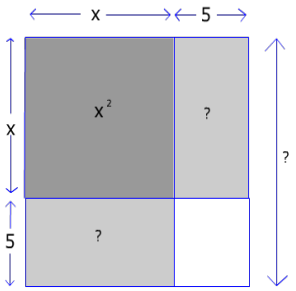
Table des matières

Chapitre 1

Algorithme

Le mot algorithme vient de la déformation du nom de AL-KHWARIZMI Muhamad Ibn Moussa (780-850) , astronome et mathématicien, membre de la maison de la sagesse à Bagdad qui fut le centre du monde scientifique de l'an 800 à l'an 1200. Il fut également l'auteur du premier traité "abrégé de calcul pour l'al-jabr et de la muqabal" . Ce mot al-jabr donna naissance au mot algèbre. A cette époque, à Bagdad, les nombres négatifs étaient inconnus. Al-Khwarizmi

utilise un algorithme à support géométrique pour déterminer la solution positive des équations du second degré lorsqu'elle se présentent sous la forme $x^2 + bx = c$ où $b > 0$ et $c > 0$
Par exemple, pour résoudre l'équation $x^2 + 10x = 39$
Il propose de tracer un carré de côté x *et de compléter par deux rectangles de dimensions* x *et la moitié de* 10 *(c'est-à-dire* 5*) pour obtenir un grand carré*
Ce grand carré a pour aire $(x^2 + 10x) + 5^2$ *c'est-à-dire* $39 + 25$ *soit* 64.



Donc il a pour côté 8 *. Il suffit alors de lui retirer* 5 *pour obtenir le côté* x *cherché :* $x = 3$

1.1 Analyse, Algorithme, Programme

Il ne faut pas confondre Analyse, Algorithme et Programme. Les algorithmes existent depuis l'antiquité (exemple : l'algorithme d'Euclide),alors que l'informatique existe depuis les années 19 mais depuis une soixantaine d'années, algorithmique et programmation vont aujourd'hui de pair.
Un algorithme est une marche à suivre pour traiter des données afin d'obtenir un résultat.
Donald KNUTH qui a créé Tex et MetaFont afin de rédiger son célèbre ouvrage The Art of Computer Programming (3 tomes) en donne la définition suivante :
"Un algorithme est un ensemble de règles qui donne un résultat à partir d'une situation donnée.Chaque étape est soigneusement définie pour que sa traduction soit claire en langage informatique et réalisable par l'ordinateur."

Un programme est le codage dans un langage de programmation d'une **analyse** qui peut comporter un ou plusieurs algorithmes. Il doit être rédigé avec des commentaires appropriés et avoir la structure formelle suivante :

```
Nom du programme .....;
  (* partie déclarations *)
    constantes
    types de variables
    variables
    sous programmes procédures
    sous programmes fonctions

  début (* du programme principal *)
(* entrée des données *)
      ..............................

    (*traitement *)
      ..............................

    (* affichage des résultats *)
      ..............................
  fin. (* du programme principal *)
```

Il faut remarquer que certaines fois, les parties traitement et affichage des résultats seront confondues car les résultats peuvent être affichés au fur et à mesure du traitement.

1.2 Une démarche de résolution

1. Etape 1 : Analyser le problème
2. Etape 2 : Concevoir un algorithme de résolution de ce problème
3. Etape 3 : Ecriture du programme qui sera exécuté par la machine.
4. Etape 4 : Tester, améliorer et maintenir ce programme

1.3 Certaines analyses sont très simples

Par exemple :
L'affichage de la surface et du périmètre d'un cercle dont l'utilisateur tape au clavier le rayon :
On sait que $P = 2 * \pi * R$ et $S = \pi * R^2$ avec $\pi \approx 3.14$

- Donnée d 'entrée : le rayon de type entier positif
- Données de sorties :
  — le périmètre de type réel
  — la surface de type réel

Voici un algorithme de résolution en Pseudo-code. :

```
Algorithme CERCLE;

  Constante PI = 3.14

  Variables RAYON de type entier

            PERIMETRE,SURFACE:de type réel ;

début

  (*entrée des données *)

    lire au clavier un nombre et le mettre dans la case mémoire RAYON

  (* traitement *)

    stocker dans la case PERIMETRE le calcul suivant:
     2 *contenu de la case PI * le contenu de la  case RAYON;

    stocker dans la case SURFACE le calcul suivant :
     le contenu de PI* le carré du contenu de la case RAYON;

  (*sortie des résultats *)

    afficher à l'écran les contenus des cases mémoires PERIMETRE et SURFACE.

fin.
```

Voici une écriture de ce programme en Turbo Pascal :

```
Program CERCLE;

  uses WinCRT;

  const PI = 3.14;

  var RAYON:integer;

      PERIMETRE,SURFACE:real;

begin

  clrscr;

  (*entrée des données *)

    write('Veuillez taper le rayon au clavier :');

    readln(RAYON);

  (* traitement *)

    PERIMETRE:=2*PI*RAYON;

    SURFACE:=PI*RAYON*RAYON;

  (*sortie des résultats *)

    writeln('Le périmètre du cercle est :',PERIMETRE);

    write('La surface de ce cercle est :',SURFACE);

end.
```

Voici une écriture de ce programme en ALGOBOX :

```
1   VARIABLES
2     RAYON EST_DU_TYPE NOMBRE
3     PERIMETRE EST_DU_TYPE NOMBRE
4     SURFACE EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     AFFICHER "(* Entree des donnees *)"
7     AFFICHER "Veillez saisir au clavier la valeur du rayon :"
8     LIRE RAYON
9     AFFICHER "Le rayon du cercle est : "
10    AFFICHER RAYON
11    AFFICHER "(* Traitement *)"
12    PERIMETRE PREND_LA_VALEUR 2*Math.PI*RAYON
13    SURFACE PREND_LA_VALEUR Math.PI*pow(RAYON,2)
14    AFFICHER "(* Sortie des resultats*)"
15    AFFICHER "Le perimetre du cercle est :"
16    AFFICHER PERIMETRE
17    AFFICHER "La surface du cercle est : "
18    AFFICHER SURFACE
19  FIN_ALGORITHME
```

Il est intéressant de faire figure dans l'algorithme et le programme sous forme de commentaires les 3 parties :

- (* entrée des données *)
- (* traitement *)
- (* affichage des résultats *)

Certaines fois, on sera obligé de traiter et d'afficher durant le traitement. Alors il n' y a aura que deux parties :

- (* entrée des données *)
- (* traitement et affichage des résultats *)

Voici une écriture de ce programme en SCILAB :

```
r = input("Entrez un clavier le rayon R = ")
p = %pi * 2 * r
s = %pi *r^2
disp("périmètre P = "+string(p))
disp("surface S = " +string(s))
```

Voici une écriture de ce programme en PYTHON :

```
>>> r = input(" Entrez au clavier le rayon r = ")
>>> pi =3.14
>>> p =  2 * pi * r
>>> s = pi * r * r
>>> print(p)
>>> print(s)
```

1.4 D'autres analyses sont plus complexes.

Il faut en tout cas :

1. Bien lire le texte du problème et bien définir le problème à traiter.
2. Bien distinguer les données en entrée : imaginer éventuellement les écrans de saisie des données
3. Bien distinguer les résultats attendus : imaginer éventuellement les écrans d'affichage des résultats
4. Trouver un traitement à effectuer sur les données en raisonnant de telle façon que le programme soit juste par construction : Aussi il faut s'efforcer de choisir **une structure de données** intéressante . Ce sont les actions sur les données qui déterminent la façon de les représenter.
   Attention, la bonne représentation des données n'est pas forçément celle qui mime la réalité !
5. Il faut décomposer le problème en des sous problèmes plus faciles à résoudre. C'est ce que l'on appelle **l'analyse descendante** qui suit le précepte machiavélique suivant "Diviser pour régner " . Ceci s'inspire du Discours de la Méthode de René Descartes : *" diviser chacune des difficultés que vous examinerez en autant de parcelles qu'il se pourra et qu'il sera requis pour mieux les résoudre "*
6. Pour cela, il faut utiliser tous les (ou une partie des) différents types de structures de contrôle :
   (a) La structure séquentielle
   (b) Les structures alternatives (si...alors... ; si ... alors ...sinon ... ;)
   (c) Les structures itératives
   - la boucle pour (for ... do begin ...end ;)
   - la boucle tant que (while ...do begin ...end ;)
   - la boucle répéter (repeat ... until ... ;)

   (d) Les sous programmes fonctions et les sous programmes procédures.

7. Ne pas oublier d'examiner la possibilité d'une **vision récursive** du problème.
   - calcul de factorielle
   - calcul de la puissance $n - ime$ d'un entier
   - tours de Hanoï
   - opérations sur les files, les listes,...
   - ...
8. Tester votre programme en n'oubliant pas qu'une bonne analyse doit tourner à la main.

Voici les 10 commandements du bon programmeur :

1 - En aucun cas tu ne taperas au clavier directement.
2 - L'énoncé tu préciseras point par point très complétement.
3 - Les programmes tu diviseras en petits blocs de traitement.
4 - C'est l'analyse descendante que tu feras à tout moment.
5 - La solution construiras pas à pas itérativement.
6 - Le problème tu résoudras quelquefois récursivement.
7 - Procédures et fonctions n'éviteras pour structure logiquement.
8 - Les commentaires utiliseras pour informer à bon escient.
9 - La bidouille tu proscriras pour progresser rapidement.
10 - En cas d'erreur, tu reprendras tout ton travail patiemment

Bon programmeur tu deviendras si tu suis ces commandements ! ! !

MICRO-JAH

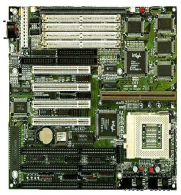
Chapitre 2

Variable, Affectation

2.1 Architecture sommaire d'un ordinateur

Un ordinateur est formé globalement de 3 parties :

1. **Un organe ou périphérique d'entrée**
   par exemple, un clavier, une souris, ...
2. **Une unité centrale ou carte-mère** formée elle-aussi de 3 parties :

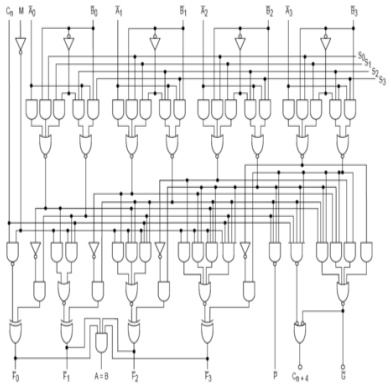


   (a) **Une mémoire centrale** séparée en deux
      i. **Mémoire vive ou RAM (Random Access Memory**
      où l'on pourra stocker des données. les cases-mémoires qui y sont seront ce que

      nous appelelrons des variables informatiques.

ii. **Mémoire morte ou ROM (Read Only memory)**
Le constructeur y stocke des routines système.

(b) **Unité arithmétique et logique**
Elle contient les circuits électroniques qui permettent de faire les opérations arithmétiques $(+, -, *, /, \dots)$ et les opérations logiques $(=, <, >, <>, \dots)$



(c) **Unité de contrôle et de commande**
pilotées par le micro-processeur impulsé par une horloge qui bat la fréquence en Hz.

3. **Un organe ou périphérique de sortie**
   par exemple un écran ou une imprimante

2.2 3 modèles d'ordinateurs

2.2.1 Les super-calculateurs

En juin 2000, le groupe informatique IBM a annoncé la construction du plus rapide et du plus gros ordinateur au monde, de la taille de deux terrains de basket, pour simuler les essais nucléaires du département américain de l'Energie.
Cet ordinateur géant, appelé ASCI White est capable d'effectuer quelque $12,3$ trillions (c'est-à-dire 12300 milliards) d'opérations par seconde.
L'ASCI White va assurer le programme de sécurité et de fiabilité de l'arsenal de l'armement nucléaire américain sans avoir recours à des essais réels.
Lors des essais dans les laboratoires d'IBM, l'ordinateur a opéré $12,3$ trillions d'opérations par seconde, dépassant de 23% le cahier des charges du département de l'Energie , selon IBM.
ASCI White marque un nouveau seuil dans la technologie informatique puisque c'est le premier ordinateur à dépasser la barre des deux chiffres en téra opérations.Il faudrait 10 millions d'années à un homme pour faire le même nombre d'opérations.
Il faudra 28 camions pour transporter les différents éléments de cet ordinateur qui sera assemblé au laboratoire californien Lawrence Livermore du département de l'énergie.
(Sources : France Antilles - Vendredi 30 juin 2000)
ASCI White était un supercalculateur au Laboratoire Lawrence Livermore en Californie .
Il s'agissait d'une grappe d'ordinateurs basée sur IBM commerciale de RS/6000 SP ordinateur. 512 de ces machines ont été reliés entre eux pour ASCI White, avec 16 processeurs par nœud et 8192 processeurs au total de 6 téraoctets de mémoire et 160 téraoctets de stockage sur disque. Malgré ces statistiques redoutables, chaque processeur est lent ici à 2006, les normes, fonctionnant à une simple 375 MHz. Par conséquent, il a été presque exclusivement utilisé pour les calculs nécessitant des dizaines de processeurs. L'ordinateur pèse 106 tonnes et consommé 3 MW d'électricité avec un autre 3 MW nécessaire pour le refroidissement. Il avait une vitesse de traitement théorique de 12,3 téraflops . Le système a fonctionné IBM système d'exploitation AIX .
ASCI White était composé de trois systèmes différents, les 512 nœuds blancs, la glace et le noeud 28 gel 68 noeuds.
Le système a été construit en Poughkeepsie, New York . Achevé en Juin 2000, il fut transporté à des installations spécialement construites en Californie et officiellement inauguré le 15 Août 2001. Performances prétendait 12.300 gigaflops, bien que cela n'a pas été atteint dans les largement acceptées LINPACK tests. Le système a coûté 110 millions de dollars.
Il a été construit comme la troisième phase de l' Initiative d'informatique stratégique accéléré (ASCI) lancé par les Etats-Unis Ministère de l'énergie et de la National Nuclear Security Administration pour construire un simulateur pour remplacer direct armes de destruction massive des tests après le moratoire sur les essais a commencé par le président George HW Bush en 1992 et

prolongée par Bill Clinton en 1993. La machine a été mise hors service début Juillet 27, 2006. *(Sources : Wikipedia - Dimanche 2 décembre 2012)*

2.2.2 Les mini-ordinateurs

Ils sont essentiellement utilisés en gestion.

2.2.3 Les micro-ordinateurs

Ce sont les ordinateurs domestiques. Au début de la micro-informatique dans les années 1970, il y avait 4 grandes familles :

1. La famille des ATARI très utilisée au début par le monde musical à cause de son interface MIDI. Atari est à l'origine une entreprise américaine pionnière dans l'industrie du jeu vidéo fondée en 1972 par Nolan Bushnell et Ted Dabney. Elle va déposer son bilan le 21 janvier 2013.

2. La famille des AMIGA :

Amiga 500 avec le moniteur et lecteur de disquette externe de marque Amiga, 1987

   L'Amiga est une famille d'ordinateurs personnels commercialisée par Commodore International entre 1985 et 1994. Dans les années 1990 il était très populaire sur la scène démo, parmi les amateurs de jeux vidéo et dans l'industrie du cinéma et de la télévision.

3. La famille des IBM-PC et compatibles articulée au début autour du micro-processeur INTEL et du système d'exploitation PC-DOS pour IBM et MS-DOS(créé par la société Microsoft de Bill GATES) Il comportait un microprocesseur Intel 8088 cadencé à 4,77 MHz et une mémoire vive de 16 ko pouvant être portée à 256 ko. Il disposait selon les modèles d'aucun, un ou deux lecteurs de disquettes 5 pouces 1/4 de 160 ko simple face, 320 ko double face, 512 ko double face double densité. Il avait 5 ports ISA, bus 8 bit, pour carte d'extension comme la carte d'extension mémoire, carte vidéo CGA. Il était équipé d'un interprète du langage BASIC Microsoft en mémoire morte et pouvait gérer4 une unité de cassette externe.


IBM PC 5150

4. La famille APPLE (crée par Steeve JOBS et) articulée au début autour du microprocesseur MOTOROLA et qui depuis une dizaine d'années s'articule autour du microprocesseur INTEL mais avec un système d'exploitation particulier MACOS (un dérivé d'UNIX)

Les deux premières familles ATARI et AMIGA ont disparu.

2.3 Affectation

Dans un langage de programmation, lorsque l'on veut faire entrer une valeur par exemple 15 dans une case-mémoire A directement par le micro-processeur. Il suffit d'écrire pour cela l'instruction $A := 15$
On réalise alors ce que l'on appelle une affectation. La case-mémoire A s'appelle une variable informatique.
On peut réaliser par exemple des affectations plus complexes du type

$$A := A + 1$$

qui est une incrémentation ou

$$A := A - 1$$

qui est une décrémentation.
Attention

A	$:=$	$A + 1$
A gauche c'est le contenant ou le nom de la case − memoire ou variable		*A droite, c'est le contenu ici le contenu de A est incremente de* 1 *et stocke dans A*

Il est fort conseillé pour suivre l'évolution des cases-mémoires(ou variables informatiques) de faire un tableau d'exécution.
En SCILAB, l'affectation se note $A = 15$ à ne pas confondre avec le booléen $A == 15$

qui est vrai si la case A contient 15 et faux sinon.

2.4 Activité

Créer un algorithme qui fait l'ordinateur permuter le contenu de 2 cases-mémoires A et B remplies au clavier auparavant par un utilisateur

2.4.1

```
Algorithme ECHANGE1;
  Variables A,B,C de même type ;
début
  (*entrée des données *)
    lire au clavier le contenu de la case mémoire A;
    lire au clavier le contenu de la case mémoire B;
  (* traitement *)
    stocker dans la case C le contenu de A;
    stocker dans la case A le contenu de B;
    stocker dans la case B le contenu de C;
  (*sortie des résultats *)
    afficher à l'écran le contenu de la case mémoire A;
    afficher à l'écran le contenu de la case mémoire B;
fin.
```

Lorsqu'on programme pour éviter d'avoir un curseur nu à l'écran lors des entrées au clavier, **dorénavant, on précédera toujours les entrées (*lire A*) par un affichage sommaire et explicatif.**

2.4.2 Codage en AlgoBox

```
1   VARIABLES
2     A EST_DU_TYPE NOMBRE
3     B EST_DU_TYPE NOMBRE
4     C EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     AFFICHER "Saisir au clavier le contenu de A :"
7     LIRE A
8     AFFICHER A
9     AFFICHER "Saisir au clavier le contenu de B :"
10    LIRE B
11    AFFICHER B
12    C PREND_LA_VALEUR A
13    A PREND_LA_VALEUR B
14    B PREND_LA_VALEUR C
15    AFFICHER "Le contenu de A est :"
16    AFFICHER A
17    AFFICHER "Le contenu de B est :"
18    AFFICHER B
19
20  FIN_ALGORITHME
```

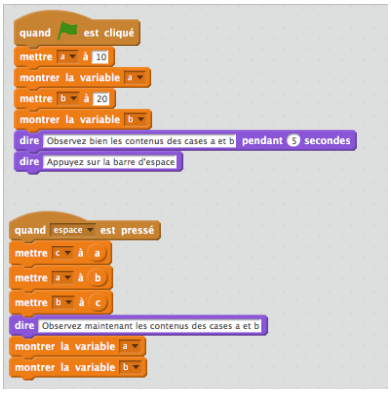
2.4.3 Codage enTurbo-Pascal

```
Program ECHANGE1;
  Var A,B,C :integer ;
begin
  (*entrée des données *)
    write(' Tapez au clavier le contenu de la case mémoire A = ');
    readln(A);
    write(' Tapez au clavier le contenu de la case mémoire B= ');
    readln(B);
  (* traitement *)
    C := A;
    A := B;
    B:= C;
  (*sortie des résultats *)
    writeln(' Le contenu de A est ', A);
    writeln(' Le contenu de B est ', B);
end.
```

	A	B	C	$ecran$
$lire(A)$	3			
$lire(B)$		8		
$C := A$			3	
$A := B$	8			
$B := C$		3		
$affiche r(A)$				8
$affiche r(B)$				3

On peut aussi permuter le contenu de deux cases contenant des noms ou des chaînes de caractères : Exemple en Turbo-Pascal, l'on saisit dans A : 'Ti Sonson' et dans B : 'Man Zosiane' à condition de déclarer A : *string* et B : *string*.
Le type **string** désignant le type chaîne de caractères.

2.4.4 Codage en Scratch



2.4.5 Codage en Scilab

```
a=input("Entrez au clavier une valeur a = ")
b=input("Entrez au clavier une valeur b = ")
c =a
a=b
b=c
disp(a,"la nouvelle valeur de a est :")
disp(b,"la nouvelle valeur de b est :")
```

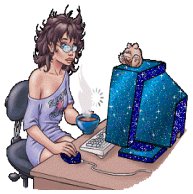
2.4.6 Codage en Python

2.5 Activité

Faites tourner à la main l'algorithme suivant qui fait l'ordinateur permuter les contenus de deux cases A et B saisis au clavier par un utilisateur.
Cette méthode n'utilise que deux cases A et B.
Cet algorithme est donc plus économe en mémoire.

2.5.1

```
Algorithme ECHANGE2;
  Variables A,B de même type ;
début
  (*entrée des données *)
    lire au clavier le contenu de la case mémoire A;
    lire au clavier le contenu de la case mémoire B;
  (* traitement *)
    stocker dans la case A le contenu de A + B;
    stocker dans la case B le contenu de A - B;
    stocker dans la case A le contenu de A - B;
  (*sortie des résultats *)
    afficher à l'écran le contenu de la case mémoire A;
    afficher à l'écran le contenu de la case mémoire B;
fin.
```



2.5.2 Codage en Algobox

```
1   VARIABLES
2     A EST_DU_TYPE NOMBRE
3     B EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     AFFICHER "Saisir au clavier le contenu de A :"
6     LIRE A
7     AFFICHER A
8     AFFICHER "Saisir au clavier le contenu de B :"
9    LIRE B
10    AFFICHER B
11    A PREND_LA_VALEUR A + B
12    B PREND_LA_VALEUR A - B
13    A PREND_LA_VALEUR A - B
14    AFFICHER "Le contenu de A est :"
15    AFFICHER A
16    AFFICHER "Le contenu de B est :"
17    AFFICHER B
18     FIN_ALGORITHME
```

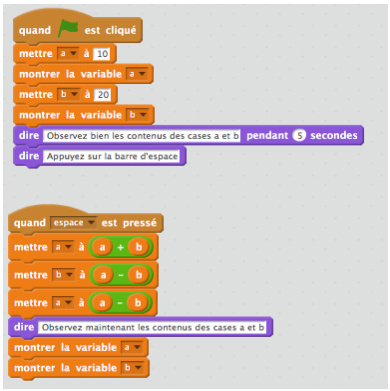
2.5.3 Codage en Turbo-Pascal

```
Program ECHANGE2
  Var A,B :integer ;
begin
  (*entrée des données *)
    write(' Tapez au clavier le contenu de la case mémoire A = ');
    readln(A);
    write(' Tapez au clavier le contenu de la case mémoire B= ');
    readln(B);
  (* traitement *)
    A := A + B;
    B := A - B;
    A := A - B;
  (*sortie des résultats *)
    writeln(' Le contenu de A est ', A);
    writeln(' Le contenu de B est ', B);
end.
```

2.5.4 Codage en Scratch

2.5.5 Codage en Scilab

```
a=input("Entrez au clavier une valeur a = ")
b=input("Entrez au clavier une valeur b = ")
a =a+b
b=a-b
a=a-b
```




```
disp(a,"la nouvelle valeur de a est :")
disp(b,"la nouvelle valeur de b est :")
```

2.5.6 Codage en Python

	A	B	$ecran$
$lire(A)$	3		
$lire(B)$		8	
$A := A + B$	11		
$B := A - B$		3	
$A := A - B$	8		
$affiche r(A)$		8	
$affiche r(B)$			3

On peut suivre de façon formelle l'évolution du prédicat suivant $\{A = a \ et \ B = b\}$ après chaque affectation :
début
$\{A = a \ et \ B = b\}$
$A := A + B$;
$\{A = a + b \ et \ B = b\}$
$B := A - B$;
$\{A = a \ et \ B = a + b - b = a\}$
$A := A - B$;
$\{A = a + b - a = b \ et \ B = a\}$
fin
Attention , ce deuxième algorithme bien que plus économe en mémoire n'est pas conseillé car il n'est pas très fiable.
Il suffit d'imaginer les effets des erreurs d'arrondis lors des calculs $A+B$; $A-B$ pour un contenu de B qui soit négligeable devant le contenu de A.
L'affectation est l'instruction qui amène les pires ennuis.
Voici un exemple d'effets surprenants dûs aux calculs approchés :

Instructions	Calcul formel	Arithmétique décimale à 2 chiffres	Arithmétique décimale à 6 chiffres
$lire(C)$	$C = 4/3$	$C = 1,33$	$C = 1,333333$
$X := 14 - 3 * C$	$X = 10$	$X = 14 - 3 \times 1,33 = 10,01$	$X = 14 - 3 \times 1,333333 = 10,000001$
$Y := 100 * (X - 10)$	$Y = 0$	$Y = 1$	$Y = 0,0001$
$affiche r(Y)$			

L'augmentation de la précision des calculs décimaux peut amener des erreurs d'arrondi.

Exemple sous Maple :

ψ^{40} est évalué directement en passant par le logarithme alors que $\frac{228826127}{2}$ et $-102334155\sqrt{52}$

```
> restart;
> psi := (1 - sqrt(5))/2;
```

$$\psi := \frac{1}{2} - \frac{1}{2}\sqrt{5}$$

```
> psi^40;
```

$$\left(\frac{1}{2} - \frac{1}{2}\sqrt{5}\right)^{40}$$

```
> expand(%);
```

$$\frac{228826127}{2} - \frac{102334155}{2}\sqrt{5}$$

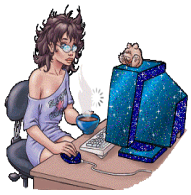
```
> evalf(%);
```

$$0.$$

```
> evalf(%%%);
```

$$4.370130127\,10^{-9}$$

sont évalués par **evalf** à la précision prévue sous Maple.



Chapitre 3

Structures de contrôle

3.1 La structure séquentielle

C'est la structure de base la plus couramment utilisée dans les algorithmes. Les instructions se suivent (séparées en général dans les langages de programmation par un " ;")

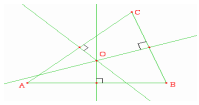
3.1.1 Exemple de séquence d'instructions

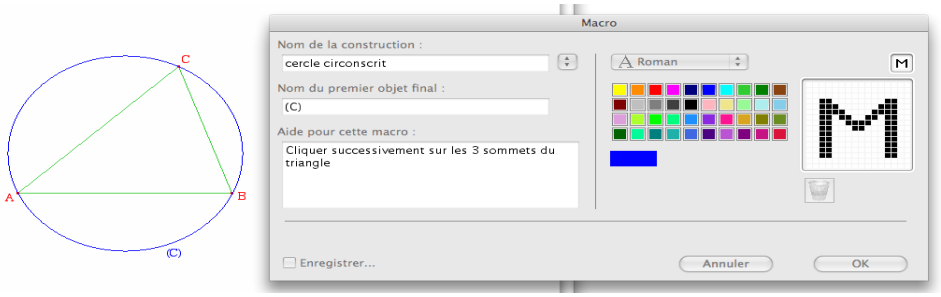
```
1 DEBUT_ALGORITHME
2     AFFICHER "(* Entree des donnees *)"
3     AFFICHER "Veillez saisir au clavier la valeur du rayon :"
4     LIRE RAYON
5     AFFICHER "Le rayon du cercle est : "
6     AFFICHER RAYON
7     AFFICHER "(* Traitement *)"
8     PERIMETRE PREND_LA_VALEUR 2*Math.PI*RAYON
9     SURFACE PREND_LA_VALEUR Math.PI*pow(RAYON,2)
10    AFFICHER "(* Sortie des resultats*)"
11    AFFICHER "Le perimetre du cercle est :"
12    AFFICHER PERIMETRE
13    AFFICHER "La surface du cercle est : "
14    AFFICHER SURFACE
15 FIN_ALGORITHME
```

3.1.2 Macro Cercle circonscrit sous Cabri Géomètre

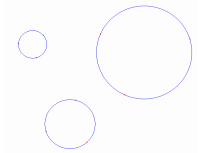
```
1 DEBUT_ALGORITHME
2     Lancer le logiciel CABRI-GEOMETRE
3     Placer un point A à l'écran
4     Placer un point B à l'écran
5     Placer un point C à l'écran
6     Tracer le segment [AB]
7     Tracer le segment [AC]
8     Tracer le segment [BC]
9     Tracer la médiatrice de [AB]
10    Tracer la médiatrice de [AC]
11    Tracer la médiatrice de [BC]
12   Tracer O le point d'intersection de deux de ces 3 médiatrices
13   Tracer (C) le cercle circonscrit au triangle ABC
14    Masquer tout ce qui a été dessiné sauf les points A,B,C et le cercle (C)
15   Sélectionner le bouton macro :
16   Prendre comme objets initiaux A,B et C
17   Prendre comme objets finaux le cercle (C)
18   Enregistrer la macro sous le nom cerclecirconscrit
19 FIN_ALGORITHME
```

On peut considérer que **la macro** cercle circonscrit **est bien un programme avec une entrée de trois données** A, B, C **sur lesquelles sont réalisées un traitement** qui fournit **en sortie**



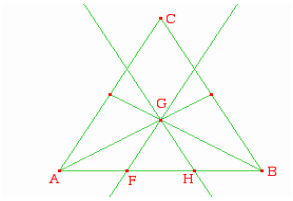


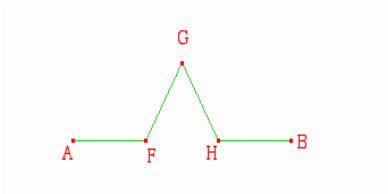
un résultat : le dessin du cercle circonscrit
Voici 3 exemples d'utilisation de cette macro CERCLE CIRCONSCRIT : On place trois points à l'écran. On charge en mémoire la macro cercle circonscrit. On sélectionne la macro puis on clique sur les 3 points, le cercle circonscrit à ces trois points se dessine alors.



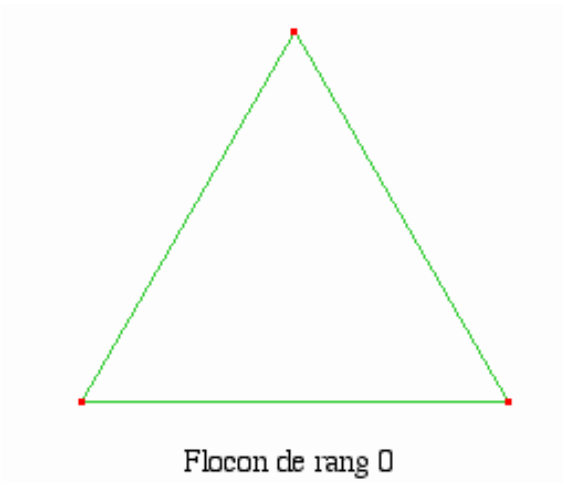
3.1.3 Macro Flocon de Von Koch sous Cabri Géomètre

```
1 DEBUT_ALGORITHME
2     Lancer le logiciel CABRI-GEOMETRE
3     Placer un point A à l'écran
4     Placer un point B à l'écran
5     Construire le point C image du point B par la rotation de centre A et d'angle de mesure 60°
6     Construire le centre de gravité G du triangle ABC
7     La parallèle à [AC] passant par G coupe le segment [AB] en F
8     La parallèle à [BC] passant par G coupe le segment [AB] en H
9     Créer les segments [AF], [FG], [GH] et [HB]
10    Masquer tout ce qui a été dessiné sauf les segments [AF],[FG],[GH] et [HB]
11   Enregistrer la macro permettant de faire correspondre aux points A et B la ligne brisée AFGHB
18 FIN_ALGORITHME
```

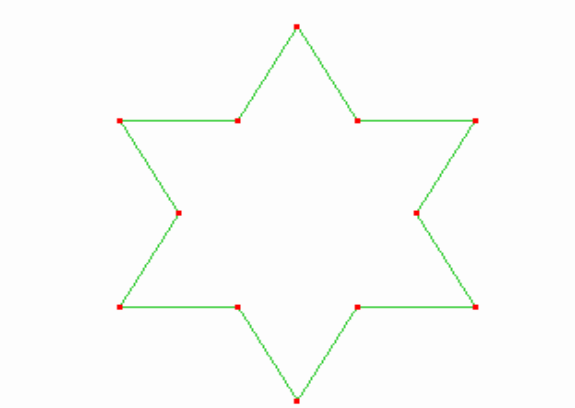




Construction de flocons à partir de la macro créée :

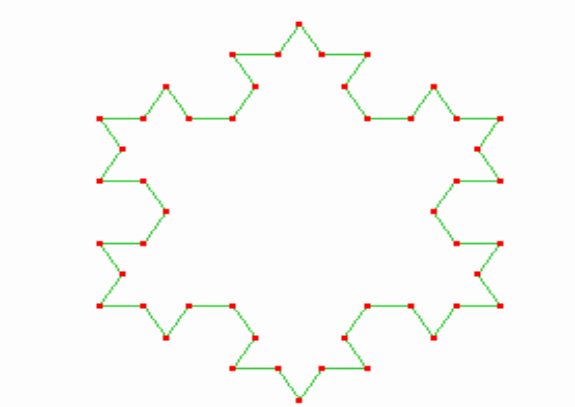


Le flocon de rang 1 est obtenu en appliquant trois fois au flocon de rang 0 la macro FLOCON



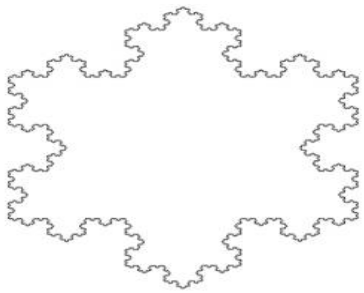
Flocon de rang 1

Le flocon de rang 2 est obtenu en appliquant trois fois au flocon de rang 1 la macro FLOCON.
Le flocon de Von Koch est la "limite" des flocons obtenus, lorsqu'on répète indéfiniment les



Flocon de rang 2

étapes mentionnées ci-dessus.



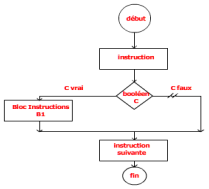
3.2 Les structures alternatives

3.2.1 L'alternative simple

C'est une instruction de la forme

```
si condition booléenne C alors début
                              bloc d'instructions B;
                           fin;
instruction suivante;
```

- Ou bien C est vraie le bloc d'instructions B suivant le "alors" est réalisé **puis** le micro-processeur passe à l'instruction suivante.
- Ou bien la condition booléenne C est fausse alors le micro-processeur passe directement à l'instruction suivante.



Algorithme de résolution de l'équation du premier degré de la forme $ax + b = 0$

Voici codé en ALGOBOX un algorithme de résolution de résolution l'un avec uniquement l'alternative simple **si .. alors ... (If ... Then ...)**

```
1   VARIABLES
2     A EST_DU_TYPE NOMBRE
3     B EST_DU_TYPE NOMBRE
4     X EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE A
7     LIRE B
8     SI (A != 0) ALORS
9       DEBUT_SI
10      X PREND_LA_VALEUR -B/A
11      AFFICHER "Une seule solution X ="
12      AFFICHER X
13      FIN_SI
14    SI (A ==0 ET B !=0) ALORS
15      DEBUT_SI
16      AFFICHER "Pas de solution"
17      FIN_SI
18    SI (A ==0 ET B ==0) ALORS
19      DEBUT_SI
20      AFFICHER "Tout réel est solution"
21      FIN_SI
22  FIN_ALGORITHME
```

3.2.2 Codage en Scilab

3.2.3 Codage en Python

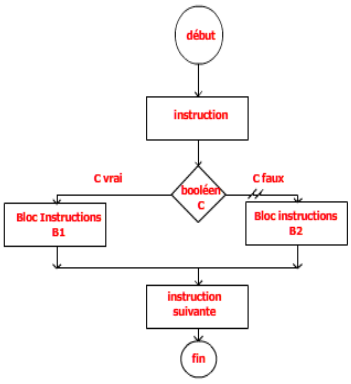
3.2.4 L'alternative complexe

C'est une instruction de la forme

```
si condition C alors début
                          bloc d'instructions B1;
                    fin
               sinon début
                          bloc d'instructions B2;
                     fin;
instruction suivante;
```

où C est une condition booléenne. De deux choses l'une :

- ou bien C est vraie : le bloc d'instructions $B1$ est alors réalisé puis le micro-processeur passe à l'instruction suivante
- ou bien C est fausse : le bloc d'instructions $B2$ est réalisé puis le micro-processeur passe à l'instruction suivante.



Exemples d'actions alternatives complexes

1. s'il fait beau je vais me promener sinon je vais au cinéma
2. la définition de la valeur absolue d'un nombre réel : si $x \geq 0$ alors $|x| = x$ sinon $|x| = -x$

Remarques

Lorsqu'on utilise des structures alternatives SI .. ALORS .... SINON ... imbriquées , il faut indenter la rédaction : placer chaque SINON sous le ALORS lui correspondant .
Exercice d'indentation : Réécrivez les deux instructions suivantes en les indentant correctement, les else étant sous les then correspondants.

1. instruction 1 :

   ```
   if C_1 then I_1 else if C_2 then I_2 else if C_3 then I_3 else I4 ;
   ```

2. instruction 2 :

   ```
   if C_1 then  if C_2 then  if C_3 then then I_1 else I_2 else I_3 else I4 ;
   ```

Corrigé

1. instruction 1 :

```
if C_1 then I_1
       else if C_2 then I_2
                   else if C_3 then I_3
                               else I4 ;
```

2. instruction 2 :

```
if C_1 then  if C_2 then  if C_3 then  I_1
                                  else I_2
                     else I_3
       else I4 ;
```

Exercice

Ecrire un algorithme de résolution de l'équation du premier degré de la forme $ax + b = 0$ en tenant compte de tous les cas, en utilisant cette fois-ci des alternatives complexes.

```
1   VARIABLES
2     A EST_DU_TYPE NOMBRE
3     B EST_DU_TYPE NOMBRE
4     X EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE A
7     LIRE B
8     SI (A != 0) ALORS
9       DEBUT_SI
10      X PREND_LA_VALEUR -B/A
11      AFFICHER "Une seule solution X ="
12      AFFICHER X
13      FIN_SI
14      SINON
15        DEBUT_SINON
16        SI (A == 0 ET B != 0 ) ALORS
17          DEBUT_SI
18          AFFICHER "Pas de solution"
19          FIN_SI
20          SINON
21            DEBUT_SINON
22            AFFICHER "Tout réel est solution"
23            FIN_SINON
24        FIN_SINON
25  FIN_ALGORITHME
```

3.2.5 Codage en Scilab

3.2.6 Codage en Python

Exercice

Ecrire un algorithme qui fait l'ordinateur afficher le plus grand entre 2 nombres A et B saisis au clavier par un utilisateur

```
1   VARIABLES
2     A EST_DU_TYPE NOMBRE
3     B EST_DU_TYPE NOMBRE
4     MAX EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE A
7     LIRE B
8     SI (A < B) ALORS
9       DEBUT_SI
10      MAX PREND_LA_VALEUR B
11      FIN_SI
12      SINON
13        DEBUT_SINON
14        MAX PREND_LA_VALEUR A
15        FIN_SINON
16    AFFICHER "Le maximum est : "
17    AFFICHER MAX
18  FIN_ALGORITHME
```

3.2.7 Codage en Scilab

3.2.8 Codage en Python

Exercice

On se place dans un environnement dans lequel sont définies 3 variables A, B et C.
Soit l'algorithme suivant écrit en pseudo-langage et décrivant un traitement :

```
si (((A < 3) ou (B > 2)) et (C>3))
   alors
           début
               B := 2;
               si A - 2*C < 0 alors C := 0;
               A := A + 2*C;
           fin
     sinon
         début
            B := 3;
            C := A + C;
          fin;
```

Expliquer dans chacun des 3 cas proposés ci-dessous ce qui se passe avec précision puis recopier et compléter sur votre copie le tableau ci-dessous :

En entrée de données on a dans	A	B	C	après traitement	A	B	C
Cas 1	1	3	4	il y a aura			
Cas 2	4	1	2	il y a aura			
Cas 3	11	3	5	il y a aura			

Voici l'écriture de cet algorithme en ALGOBOX :

```
VARIABLES
  A EST_DU_TYPE NOMBRE
  B EST_DU_TYPE NOMBRE
  C EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE A
  LIRE B
  LIRE C
  SI (((A<3) OU (B>2)) ET (C>3)) ALORS
    DEBUT_SI
    B PREND_LA_VALEUR 2
    SI (A-2*C<0) ALORS
      DEBUT_SI
      C PREND_LA_VALEUR 0
      FIN_SI
    A PREND_LA_VALEUR A+2*C
    FIN_SI
    SINON
      DEBUT_SINON
      B PREND_LA_VALEUR 3
      C PREND_LA_VALEUR A+C
      FIN_SINON
  AFFICHER A
  AFFICHER B
  AFFICHER C
FIN_ALGORITHME
```

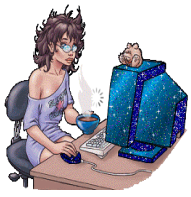
3.2.9 Codage en Scilab

3.2.10 Codage en Python

Programmes de Résolution de $ax^2 + bx + c = 0$

En ALGOBOX :

```
1   VARIABLES
2     X1 EST_DU_TYPE NOMBRE
3     X2 EST_DU_TYPE NOMBRE
4     X0 EST_DU_TYPE NOMBRE
5     a EST_DU_TYPE NOMBRE
6     b EST_DU_TYPE NOMBRE
7     c EST_DU_TYPE NOMBRE
8     DELTA EST_DU_TYPE NOMBRE
9   DEBUT_ALGORITHME
10    LIRE a
11    LIRE b
12    LIRE c
13    DELTA PREND_LA_VALEUR b*b - 4*a*c
14    SI (DELTA > 0) ALORS
15      DEBUT_SI
16      AFFICHER "Deux solutions X1 et X2"
17      X1 PREND_LA_VALEUR (-b + sqrt(DELTA))/(2*a)
18      X2 PREND_LA_VALEUR (-b -sqrt(DELTA))/(2*a)
19      AFFICHER X1
20      AFFICHER X2
21      FIN_SI
22      SINON
23        DEBUT_SINON
24        SI (DELTA <0) ALORS
25          DEBUT_SI
26          AFFICHER "Pas de solutions"
27          FIN_SI
28          SINON
29            DEBUT_SINON
30            AFFICHER "Une seule solution"
31            X0 PREND_LA_VALEUR -b/(2*a)
32            AFFICHER X0
33            FIN_SINON
34        FIN_SINON
35  FIN_ALGORITHME
```



En Turbo-Pascal

```
 program second_degre;
uses WinCrt;
var A,B,C,DELTA,X1,X2 : real;
begin
(* entrée des données *)
     write('Tapez A=');
     readln(A);
     write('Tapez B=');
     readln(B);
     write('Tapez C=');
     readln(C);
(* traitement et affichage des résultats *)
   DELTA := B*B-4*A*C;
   if DELTA < 0
      then writeln('Pas de solution ')
      else if DELTA = 0
             then begin
                   X1 := -B/(2*A);
                   writeln(' Une seule solution est X !',X1);
                  end
             else begin
                   X1 := (- B - sqrt(DELTA))/(2*A);
                   X2 := (- B + sqrt(DELTA))/(2*A);
                   writeln(' Deux solutions, la première est X1 = ',X1);
                   writeln(' l''autre solution est X2 =',X2);
                   end;
end.
```

En langage calculatrice Casio :

```
 "trinome du second degre"
 "A" ->A :  "B" ->B :  "C" ->C : B*B - 4*A*C -> D
 "Delta =" : D icône triangle noir
 D = 0 implique Goto 0
 D > 0 implique Goto 1
 D < 0 implique Goto 2
 Lbl 0
  "Racine double X = " : B / 2*A : Goto 3
Lbl 1
(-B - racine carrée (D))/(2*A) -> M
(-B + racine carrée (D))/(2*A) -> N
"2 Racines simples " : M icône triangle noir
N icône triangle noir
Goto 3
Lbl 2
"Pas de racines"
Lbl 3
"Fin"
```

En langage calculatrice TI

```
TRI :
input A :
input B :
input C :
B * B - 4 * A * C -> D :
disp "DELTA =" :
disp D :
if D < 0 : goto 1 :
if D = 0 : goto 2 :
goto 3 :
LBL 1 : disp "Pas de solution réelle " : goto END :
LBL 2 : - B / (2 * A) -> X : disp X : goto END
LBL 3 : (- B + racinecarrée(D))/(2 * A) -> X1 : (- B - racinecarrée(D))/(2 * A) -> X2 :
disp X1 : disp X2 : goto END :
END :
disp "Fin" :
```

3.2.11 Codage en Scilab

3.2.12 Codage en Python

Exercice

Ecrire un algorithme arrondissant une note décimale au demi près.

```
variable NOTE, NOTE_ARRONDIE : réel;
début
    lire(NOTE);
    DIFFERENCE := NOTE - partieentière(NOTE);
    si DIFFERENCE = 0
         alors NOTE_ARRONDIE := NOTE
         sinon si DIFFERENCE <= 0.5
                   alors NOTE_ARRONDIE := partieentiere(NOTE) + 0.5
                   sinon NOTE_ARRONDIE := partieentiere(NOTE + 1)
               finsi
    finsi;
    afficher(NOTE_ARRONDIE);
 fin.
```

3.2.13 Codage en Scilab

3.2.14 Codage en Python

Obtention du baccalauréat

Ecrire un algorithme qui saisit la moyenne d'un candidat et qui lui affiche une des 3 réponses suivantes :

- reçu directement lorsque la moyenne obtenue après le premier groupe est au dessus ou égale à 10
- oral de rattrapage lorsque la moyenne obtenue après le premier groupe est entre 8 et 10
- recalé lorsque la moyenne obtenue après le premier groupe est strictement inférieure à 8

```
1   VARIABLES
2     M EST_DU_TYPE NOMBRE
3   DEBUT_ALGORITHME
4     LIRE M
5     SI (M<8) ALORS
6       DEBUT_SI
7       AFFICHER "recale"
8       FIN_SI
9       SINON
10        DEBUT_SINON
11        SI (M>=10) ALORS
12          DEBUT_SI
13          AFFICHER "recu"
14          FIN_SI
15          SINON
```

```
16            DEBUT_SINON
17            AFFICHER "oral"
18            FIN_SINON
19        FIN_SINON
20  FIN_ALGORITHME
```

3.2.15 Codage en Scilab

3.2.16 Codage en Python

3.2.17 Exercice : Bac avec mention

Affiner l'algorithme précédent pour afficher les mentions éventuelles.

- mention très bien lorsque la moyenne obtenue après le premier groupe est au dessus ou égale à 16
- mention bien lorsque la moyenne obtenue après le premier groupe est entre 14 et 16
- mention assez bien lorsque la moyenne obtenue après le premier groupe est entre 12 et 14

```
1   VARIABLES
2     M EST_DU_TYPE NOMBRE
3   DEBUT_ALGORITHME
4     LIRE M
5     TANT_QUE (M<0 OU M > 20) FAIRE
6       DEBUT_TANT_QUE
7       AFFICHER "Erreur de frappe Tapez une note entre 0 et 20"
8       LIRE M
9       FIN_TANT_QUE
10    SI (M<8) ALORS
11      DEBUT_SI
12      AFFICHER "recale"
13      FIN_SI
14      SINON
15        DEBUT_SINON
16        SI (M>=10) ALORS
17          DEBUT_SI
18          AFFICHER "reçu"
19          SI (M >=16) ALORS
20            DEBUT_SI
21            AFFICHER "Mention Très Bien"
22            FIN_SI
23            SINON
24              DEBUT_SINON
25              SI (M >=14) ALORS
26                DEBUT_SI
27                AFFICHER "Mention Bien"
28                FIN_SI
29                SINON
30                  DEBUT_SINON
31                  SI (M >=12) ALORS
32                    DEBUT_SI
33                    AFFICHER "Mention Assez Bien"
34                    FIN_SI
35                  FIN_SINON
36              FIN_SINON
37          FIN_SI
38          SINON
39            DEBUT_SINON
40            AFFICHER "oral"
41            AFFICHER "Donnez nous votre moyenne d'Oral :"
42            LIRE M
```

```
43            SI (M<10) ALORS
44              DEBUT_SI
45              AFFICHER "Revenez l'an prochain"
46              FIN_SI
47              SINON
48                DEBUT_SINON
49                AFFICHER "Champagne : vous l'avez enfin eu !"
50                FIN_SINON
51            FIN_SINON
52        FIN_SINON
53  FIN_ALGORITHME
```

3.2.18 Codage en Scilab

3.2.19 Codage en Python

3.2.20 L'alternative multiple SELON ...

Exercice de Choix multiple

Des enfants âgés de 6 à 13 ans se présentent à l'accueil d'un club sportif. Ils voudraient savoir dans quelle catégorie faire leur licence. Ecrire un algorithme qui les informe de leur catégorie sachant que :

1. "poussin" de 6 à 7 ans
2. "pupilles" de 8 à 9 ans
3. "minimes" de 10 à 11 ans
4. "cadets" de 12 à 13 ans

Cet exercice est rédigé en ALGOBOX en utilisant 3 fois la structure alternative complexe si .. alors ... sinon ...

```
var AGE : 6..14;
début
     lire(AGE);
     si AGE < 6
          alors afficher("pas de catégorie")
          sinon si AGE <= 7
                    alors afficher ("poussin")
                    sinon si AGE <= 9
                              alors afficher("pupilles")
                              sinon si AGE <=11
                                        alors afficher("minimes")
                                        sinon afficher("cadets")
                                   finsi
                         finsi
               finsi
      finsi;
 fin.
```

3.2.21 Instructions de choix multiples

Certains langages (par exemple Turbo Pascal ou Caml) disposent d'instructions gérant une alternative multiple.

```
case of
      CHOIX = 1 begin bloc d'instructions1 end;
      CHOIX = 2 begin bloc d'instructions2 end;
      CHOIX = 3 begin bloc d'instructions3 end;
      ..............
      CHOIX = n begin bloc d'instructionsn end;
      else
                     begin bloc d'instructions  end;
endcase;
```

Utilisation du case of en Turbo Pascal

```
program licences_sportives;
uses WinCrt;
var AGE : 6..14;
begin
(* entrée des données *)
     write('Tapez l''âge du joueur licencié : ');
     readln(AGE);

(* traitement et affichage des résultats *)
   case AGE of
              6..7 : begin writeln('C''est un Poussin') end;
              8..9 : begin writeln('C''est un Pupille') end;
              10..11 : begin writeln('C''est un Minime') end;
              12..13 : begin writeln('C''est un Cadet') end;
              14..15 : begin writeln('C''est un junior') end;
              end;
end.
```

3.3 Les structures itératives

Pour bien cerner une structure répétitive appelée aussi structure itérative ou boucle, il faut se poser uniquement une seule question : **le nombre de répétitions est-il connu ou inconnu ?**

- Lorsque le nombre de répétitions est connu, on est en présence d'une structure POUR (ou FOR)
- Lorsque le nombre de répétitions est connu, on peut utiliser :
  — soit une structure TANT QUE (ou WHILE)
  — soit une structure REPETER ... JUSQU'A (ou REPEAT ... UNTIL ...)

3.3.1 la boucle pour (POUR... FAIRE début ...fin ;)

Elle a la forme suivante

```
POUR variable de boucle VARIANT d'une valeur initiale à une valeur finale
                FAIRE
                     DEBUT
                          Bloc d'instructions
                     FIN POUR;
```

Exemples

En Turbo-Pascal, elle prend les formes suivantes :

-
```
for I := 1 to 15  do begin
                                    bloc instructions ;
                              end;
```

- Pour un compte à rebours
```
for I := 15 downto 1  do begin
                                     bloc instructions ;
                                end;
```

-

En Maple, elle s'écrit avec la possibilité de mettre un pas (ici un pas de 2) :

```
for I from 1 to 15 by 2 do
                          bloc instructions ;
                        od;
```

En algobox , voici un algorithme affichant les quinze premiers entiers naturels non nuls.

```
1  VARIABLES
2    I EST_DU_TYPE NOMBRE
3  DEBUT_ALGORITHME
4    POUR I ALLANT_DE 1 A 15
5      DEBUT_POUR
6      AFFICHER I
7      FIN_POUR
8  FIN_ALGORITHME
```

Attention, Ne modifiez pas la variable de boucle I dans le corps d'une boucle POUR

3.3.2 Codage en Scilab

3.3.3 Codage en Python

Exercice - Niveau 2de

Ecrire un algorithme affichant la somme des n premiers entiers naturels non nuls

```
1   VARIABLES
2     N EST_DU_TYPE NOMBRE
3     I EST_DU_TYPE NOMBRE
4     SOMME EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     AFFICHER "Entrée des données"
7     LIRE N
8     AFFICHER "Traitement"
9     SOMME PREND_LA_VALEUR 0
10    POUR I ALLANT_DE 1 A N
11      DEBUT_POUR
12      SOMME PREND_LA_VALEUR SOMME + I
13      FIN_POUR
14    AFFICHER "Sortie des Résultats"
15    AFFICHER "La somme est :  "
16    AFFICHER SOMME
17  FIN_ALGORITHME
```

3.3.4 Codage en Scilab

3.3.5 Codage en Python

3.3.6 Codage en Caml

```
let somme n =
     let s = ref 0 in
                          for i = 1 to n
```

```
                        do
                           s := !s + i
                        done;
                        !s
   ;;
```

Exercice - Niveau 1ères S

Ecrire un algorithme affichant le produit des n premiers entiers naturels non nuls noté $n!$ et appelé factorielle de n

```
1   VARIABLES
2     N EST_DU_TYPE NOMBRE
3     I EST_DU_TYPE NOMBRE
4     PRODUIT EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     AFFICHER "Entrée des données"
7     LIRE N
8     AFFICHER "Traitement"
9     PRODUIT PREND_LA_VALEUR 1
10    POUR I ALLANT_DE 1 A N
11      DEBUT_POUR
12      PRODUIT PREND_LA_VALEUR PRODUIT * I
13      FIN_POUR
14    AFFICHER "Sortie des Résultats"
15    AFFICHER "Le produit est :  "
16    AFFICHER PRODUIT
17  FIN_ALGORITHME
```

3.3.7 Codage en Scilab

3.3.8 Codage en Python

3.3.9 Codage en Caml

```
let fact n =
     let p = ref 1 in
                        for k = 1 to n
                          do
                             p := !p * k
                          done;
                          !p
   ;;
```

Exercice

Ecrire un algorithme qui :

- permet à un utilisateur de saisir au clavier une liste de n notes affectées chacune d'un coefficient
- affiche la moyenne arrondie de ces n notes

On utilisera la structure de données Tableau ou liste.

Programme ALGOBOX :

```
1   VARIABLES
2     NOTE EST_DU_TYPE LISTE
3     COEFF EST_DU_TYPE LISTE
4     I EST_DU_TYPE NOMBRE
5     MOYENNE EST_DU_TYPE NOMBRE
6     N EST_DU_TYPE NOMBRE
7     SOMMECOEFF EST_DU_TYPE NOMBRE
8     TOTALNOTES EST_DU_TYPE NOMBRE
9   DEBUT_ALGORITHME
10    AFFICHER "Entrée des données"
11    LIRE N
12    POUR I ALLANT_DE 1 A N
13      DEBUT_POUR
14      LIRE NOTE[I]
15      LIRE COEFF[I]
16      FIN_POUR
17    AFFICHER "Traitement"
18    TOTALNOTES PREND_LA_VALEUR 0
19    SOMMECOEFF PREND_LA_VALEUR 0
20    POUR I ALLANT_DE 1 A N
21      DEBUT_POUR
22      TOTALNOTES PREND_LA_VALEUR TOTALNOTES + NOTE[I]*COEFF[I]
23      SOMMECOEFF PREND_LA_VALEUR SOMMECOEFF + COEFF[I]
24      FIN_POUR
25    AFFICHER "Sortie des résultats"
26    MOYENNE PREND_LA_VALEUR TOTALNOTES/SOMMECOEFF
27    AFFICHER "La moyenne coefficientée est : "
28    AFFICHER MOYENNE
29  FIN_ALGORITHME
```

3.3.10 Codage en Scilab

3.3.11 Codage en Python

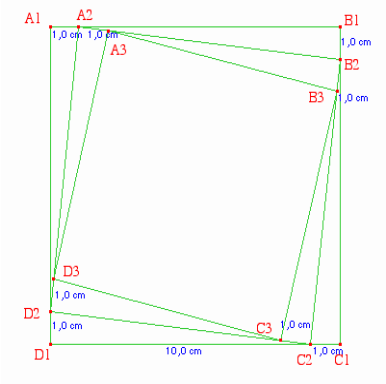
Exercice : La poursuite infernale - extrait Hypercube Mars 95 - Niveau terminale S

4 copains A_1, B_1, C_1 et D_1 sont situés aux quatre sommets d'un carré de 10 m de côté.
Ils s'amusent à se poursuivre en courant tous à la même vitesse mais en faisant des enjambées de 1 m.
Avant chaque enjambée, chacun observe la position de sa cible et se dirige vers elle par le court chemin :
A_1 court après B_1,B_1 court après C_1,C_1 court après D_1,D_1 court après $A_1 \cdots$

1. Faites un dessin.
2. Démontrer que $A_2B_2C_2D_2$,$A_3B_3C_3D_3$, $A_4B_4C_4D_4$ sont des carrés.
3. Quel est leur centre ? Justifier.
4. Calculer A_2B_2, A_3B_3.
5. En écrivant un algorithme,
   (a) Remplir le tableau suivant :

A_1B_1	A_2B_2	A_3B_3	A_4B_4	A_5B_5	A_6B_6	A_7B_7	A_8B_8	A_9B_9	$A_{10}B_{10}$	$A_{11}B_{11}$	$A_{12}B_{12}$	$A_{13}B_{13}$

   (b) Conjecturer sur la situation suivante : les 4 poursuivants rattraperont-ils leur cible ?



Corrigé

1. Les figures $A_2B_2C_2D_2$,$A_3B_3C_3D_3$, $A_4B_4C_4D_4, \cdots$ sont des carrés de même centre
   (a) Le quadrilatère $A_2B_2C_2D_2$ est un carré car c'est un losange ayant un angle droit.
      i. Les 4 côtés $A_2B_2, B_2C_2, C_2D_2, D_2A_2$ ont même longueur en utilisant le théorème de Pythagore , leur longueur vaut $\sqrt{9^2+1^2}$
      ii. L'angle $\widehat{D_2A_2B_2}$ est droit.

      - les angles géométriques $\widehat{A_1D_2A_2}$ et $\widehat{A_1A_2D_2}$ sont complémentaires.
      - les angles géométriques $\widehat{A_1D_2A_2}$ et $\widehat{B_2A_2B_1}$ sont égaux car ils ont même tangente $\dfrac{1}{10}$.
      - $\pi = \widehat{A_1A_2D_2} + \widehat{D_2A_2B_2} + \widehat{B_2A_2B_1} = \widehat{D_2A_2B_2} + \dfrac{\pi}{2}$
      - d'où $\widehat{D_2A_2B_2} = \dfrac{\pi}{2}$

   (b) Idem pour $A_3B_3C_3D_3$, $A_4B_4C_4D_4, \cdots$
   (c) Le quadrilatère $A_2B_2C_2D_2$ est un carré de même centre O que la carré initial $A_1B_1C_1D_1$. En effet, le quadrilatère $A_1A_2C_1C_2$ a deux côtés parallèles et de même longueur donc est un parallélogramme. ses diagonales vont se couper en leur milieu. Le milieu de $[A_2C_2]$ est le milieu de $[A_1C_1]$ qui est aussi le milieu de $[B_1D_1]$ donc O le centre du carré initial.
   (d) Le triangle $A_2B_1B_2$ est rectangle en B_1 donc d'après le théorème de Pythagore, l'on a :
   $A_2B_2^2 = A_2B_1^2 + B_1B_2^2 = (10-1)^2 + 1^2 = 82$ donc $A_2B_2 = \sqrt{82} \approx 9,055$ Le triangle $A_3B_2B_3$ est rectangle en B_2 donc d'après le théorème de Pythagore, l'on a :
   $A_3B_3^2 = A_3B_2^2 + B_2B_3^2 = (\sqrt{82}-1)^2 + 1^2$ donc $A_3B_3 \approx 8,117$

   On peut donc simuler les longueurs successives A_4B_4, $A_5B_5, \cdots$ avec l'algorithme :

```
Prendre le côté précédent
Lui enlever 1
Elever au carré
Ajouter 1 (le carré de 1)
Prendre la racine carrée
```

   Voici l'algorithme en Algobox :

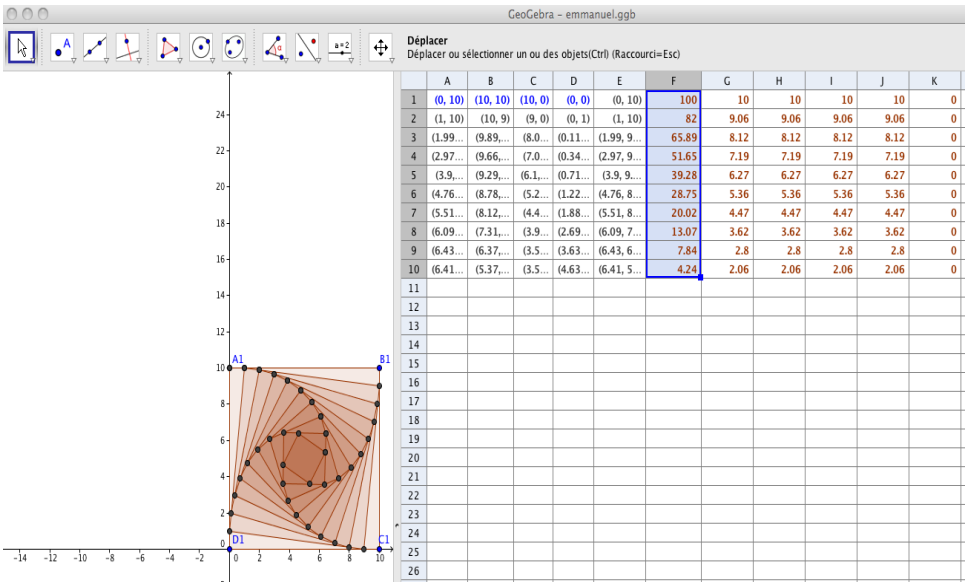
```
1   VARIABLES
2     u EST_DU_TYPE NOMBRE
3     k EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     u PREND_LA_VALEUR 10
6     POUR k ALLANT_DE 1 A 100
7       DEBUT_POUR
8       u PREND_LA_VALEUR u-1
9       u PREND_LA_VALEUR u*u
10      u PREND_LA_VALEUR u+1
11      u PREND_LA_VALEUR sqrt(u)
12      AFFICHER u
13      FIN_POUR
14  FIN_ALGORITHME
```

A_2B_2	A_3B_3	A_4B_4	A_5B_5	A_6B_6	A_7B_7	A_8B_8	A_9B_9	$A_{10}B_{10}$	$A_{11}B_{11}$	$A_{12}B_{12}$	$A_{13}B_{13}$	$A_{14}B_{14}$
$9,055$	$8,117$	$7,187$	$6,267$	$5,361$	$4,474$	$3,615$	$2,8$	$2,059$	$1,456$	$1,090$	$1,004$	1

A la longue, les 4 copains ne se rencontreront jamais, restant à une distance 1 m de leur cible.
On peut l'expliquer

- soit en remarquant qu'une fois que u prend la valeur 1 dans la boucle u va toujours garder cette valeur.
- soit en réfléchissant à la situation de poursuite : la longueur de l'enjambée est alors égale au côté du carré. La situation ne va plus évoluer.

On peut illustrer cette situation en utilisant le logiciel Geogebra et l'option Tableur dans le menu Affichage



3.3.12 la boucle TANT QUE

Elle a la forme suivante

```
TANT QUE condition booléenne
                FAIRE
                     DEBUT
                          Bloc d'instructions
                     FIN TANT QUE;
instruction suivante;
```

Si la condition booléenne est vraie le bloc d'instructions est réalisé sinon le processeur passe à l'instruction suivant la boucle.

Exemples

(while ...do begin ...end ;)

3.3.13 la boucle REPETER

Elle a la forme suivante

```
REPETER
         Bloc d'instructions
JUSQU'A condition booléenne;
instruction suivante;
```

Le bloc d'instructions est réalisé au moins une fois et ceci jusqu'à ce que la condition booléenne devienne vraie.

Exemples

(repeat . . . until ... ;)

> ATTENTION !
> Lorsqu'on utilise une structure TANT QUE ou REPETER 2 précautions sont indispensables.
>
> 1. Ne pas oublier l'initialisation avant d'entrer dans le corps de la boucle
> 2. Dans le corps de la boucle , l'état de la machine doit évoluer vers la sortie de boucle. La condition booléenne de sortie doit changer d'état à un moment sinon la boucle tournera indéfiniment.

Exercice Niveau 1ère : Le juste prix par dichotomie

Un utilisateur a droit au maximum à 7 essais pour deviner le juste prix s d'un article. Ce prix est compris entre 10 euros et 100 euros. Ecrire un algorithme lui permettant de retrouver ce prix.
Corrigé étudiants Allan Deschamps et Jérôme Mercan EC1 2011-12 :

```
1   VARIABLES
2     n EST_DU_TYPE NOMBRE
3     s EST_DU_TYPE NOMBRE
4     nbessais EST_DU_TYPE NOMBRE
5     p EST_DU_TYPE NOMBRE
6   DEBUT_ALGORITHME
7     AFFICHER "Le nombre maximum d'essais est 7 "
8     nbessais PREND_LA_VALEUR 7
9     s PREND_LA_VALEUR floor(91*random())+10
10    AFFICHER "J'ai choisi un nombre entre 10 et 100. Devine-le"
11    TANT_QUE (nbessais > 0) FAIRE
12      DEBUT_TANT_QUE
13      LIRE n
14      SI (n < s) ALORS
15        DEBUT_SI
16        AFFICHER "Trop peu"
17        nbessais PREND_LA_VALEUR nbessais - 1
18        FIN_SI
19        SINON
20          DEBUT_SINON
21          SI (n > s) ALORS
22            DEBUT_SI
23            AFFICHER "Trop grand !!!"
24            nbessais PREND_LA_VALEUR nbessais - 1
25            FIN_SI
26            SINON
27              DEBUT_SINON
28              p PREND_LA_VALEUR 7 - nbessais
29              nbessais PREND_LA_VALEUR -1
30              FIN_SINON
31          FIN_SINON
32      FIN_TANT_QUE
33    SI (nbessais==- 1) ALORS
34      DEBUT_SI
35      AFFICHER "c'est fini vous avez gagné en "
36      AFFICHER p
37      AFFICHER " essais  "
38      FIN_SI
39      SINON
40        DEBUT_SINON
41        AFFICHER "Vous avez perdu !!!"
42        AFFICHER "Le nombre à trouver était : "
43        AFFICHER s
44        FIN_SINON
45  FIN_ALGORITHME
```

Corrigé Kabrit bwa :

```
1   VARIABLES
2     essai EST_DU_TYPE NOMBRE
3     n EST_DU_TYPE NOMBRE
4     s EST_DU_TYPE NOMBRE
5     compteur EST_DU_TYPE NOMBRE
6     nbessais EST_DU_TYPE NOMBRE
7   DEBUT_ALGORITHME
8     AFFICHER "Nombre maximum d'essais inférieur à 7 ?"
9     LIRE nbessais
10    TANT_QUE (nbessais<1 OU nbessais >7) FAIRE
11      DEBUT_TANT_QUE
12      LIRE nbessais
13      FIN_TANT_QUE
14    AFFICHER "Nombre maximum d'essais : "
15    AFFICHER nbessais
16    s PREND_LA_VALEUR floor(91*random())+10
17    AFFICHER "J'ai choisi un nombre entre 10 et 100. Devine-le"
18    essai PREND_LA_VALEUR 0
19    TANT_QUE (essai<=nbessais) FAIRE
20      DEBUT_TANT_QUE
21      AFFICHER "Il vous reste   "
22      compteur PREND_LA_VALEUR nbessais - essai
23      AFFICHER compteur
24      AFFICHER "   essais"
25      LIRE n
26      SI (n>s) ALORS
27        DEBUT_SI
28        AFFICHER "trop"
29        FIN_SI
30        SINON
31          DEBUT_SINON
32          SI (n<s) ALORS
33            DEBUT_SI
34            AFFICHER "trop peu"
35            FIN_SI
36            SINON
37              DEBUT_SINON
38              AFFICHER "gagne en  "
39              AFFICHER essai
40              AFFICHER " essais"
41              essai PREND_LA_VALEUR nbessais + 2
42              FIN_SINON
43          FIN_SINON
44      essai PREND_LA_VALEUR essai+1
45      FIN_TANT_QUE
46    SI (essai==nbessais + 1) ALORS
47      DEBUT_SI
48      AFFICHER "c'est fini vous avez perdu, le nombre qu'il fallait trouver est : "
49      AFFICHER s
50      FIN_SI
51  FIN_ALGORITHME
```

3.3.14 Codage en Scilab

3.3.15 Codage en Python

Exercice - Niveau 2de

Ecrire un algorithme simulant le tirage du LOTO en utilisant une boucle "TANT QUE". On doit choisir 5 nombres parmi les nombres de 1 à 49.
Algorithme 1 :

```
variables
     COMPTEUR : 1..6;
     NUMERO : 1..49;
     SORTIE : tableau[1.. NUMERO] de booléens;
début algorithme

(* il faut initialiser le tableau de vérification de sortie des 49 nombres *)

     pour NUMERO variant de 1 à 49 faire SORTIE[NUMERO] := faux ;

 (* il faut effectuer ensuite le tirage des 5 numéros *)

     COMPTEUR := 0;
     tant que COMPTEUR < 6
     faire début
                  COMPTEUR := COMPTEUR + 1;
                  CHOIX := random(49) + 1;
                  si SORTIE[CHOIX] = faux alors SORTIE[CHOIX] := vrai
                                               sinon COMPTEUR := COMPTEUR - 1 ;
              fin;

  (* affichage des 5 numéros *)

       pour NUMERO variant de 1 à 49
       faire
             début
                  si SORTIE[NUMERO] = vrai alors afficher(NUMERO);
             fin ;

fin algorithme
```

On peut améliorer l'affichage des 5 numéros en ne parcourant pas systématiquement toutes les 49 cases du tableau SORTIE

3.3.16 Codage en Scilab

3.3.17 Codage en Python

Algorithme 2 :

```
variables
     COMPTEUR, COMPTEURNUMEOAFFICHE : 1..6;
     NUMERO : 1..49;
     SORTIE : tableau[1.. NUMERO] de booléens;
début algorithme
(* il faut initialiser le tableau de vérification de sortie des 49 nombres *)
     pour NUMERO variant de 1 à 49 faire SORTIE[NUMERO] := faux ;
 (* il faut effectuer ensuite le tirage des 5 numéros *)
     COMPTEUR := 0;
     tant que COMPTEUR < 6
     faire début  COMPTEUR := COMPTEUR + 1;
                       CHOIX := random(49) + 1;
                       si SORTIE[CHOIX] = faux alors SORTIE[CHOIX] := vrai
                                                sinon COMPTEUR := COMPTEUR - 1 ;
               fin;
  (* affichage des 5 numéros *)
     COMPTEURNUMEROAFFICHE :=0;
     NUMERO := 0;
     tant que COMPTEURNUMEROAFFICHE < 6
     faire debut
          incrementer(NUMERO);
          si SORTIE[NUMERO] = vrai alors debut
                                          afficher(NUMERO);
                                          incrémenter(COMPTEURNUMEROAFFICHE);
                                         fin
fin algorithme
```

3.3.18 Codage en Scilab

3.3.19 Codage en Python

hyperbole 2de p258 n° 54

Dans un certain pays , on recommande aux couples souhaitant fonder une famille d'avoir des enfants jusqu'à ce que les deux sexes soient représentés

1. Donner une façon de simuler la composition d'une famille obéissant à cette politique

```
1   VARIABLES
2     enfant1 EST_DU_TYPE NOMBRE
3     nouvelenfant EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     enfant1 PREND_LA_VALEUR floor(2*random())
6     SI (enfant1==0) ALORS
7       DEBUT_SI
8       AFFICHER "Le premier enfant est un garçon"
9       FIN_SI
10      SINON
11        DEBUT_SINON
12        AFFICHER "Le premier enfant est une fille"
13        FIN_SINON
14    nouvelenfant PREND_LA_VALEUR enfant1
15    TANT_QUE (nouvelenfant ==enfant1) FAIRE
16      DEBUT_TANT_QUE
17      nouvelenfant PREND_LA_VALEUR floor(2*random())
18      SI (nouvelenfant==0) ALORS
19        DEBUT_SI
20        AFFICHER "l'enfant suivant est un garçon"
21        FIN_SI
22        SINON
23          DEBUT_SINON
24          AFFICHER "l'enfant suivant est une fille"
25          FIN_SINON
26      FIN_TANT_QUE
27    AFFICHER "Stop !! !Plus de naissances !!!"
28  FIN_ALGORITHME
```

3.3.20 Codage en Scilab

3.3.21 Codage en Python

2. Réaliser 100 simulations et noter à chaque fois le nombre d'enfants de la famille puis calculer pour l'échantillon obtenu, le nombre moyen d'enfants par famille

```
1  VARIABLES
2    enfant1 EST_DU_TYPE NOMBRE
3    nouvelenfant EST_DU_TYPE NOMBRE
4    I EST_DU_TYPE NOMBRE
5    NBENFANTS EST_DU_TYPE NOMBRE
6    MOYENNE EST_DU_TYPE NOMBRE
7  DEBUT_ALGORITHME
8    NBENFANTS PREND_LA_VALEUR 0
9    POUR I ALLANT_DE 1 A 100
10     DEBUT_POUR
11     enfant1 PREND_LA_VALEUR floor(2*random())
12     SI (enfant1==0) ALORS
13       DEBUT_SI
14       AFFICHER "Le premier enfant est un garçon"
15       FIN_SI
16       SINON
17         DEBUT_SINON
18         AFFICHER "Le premier enfant est une fille"
19         FIN_SINON
20     NBENFANTS PREND_LA_VALEUR NBENFANTS + 1
21     nouvelenfant PREND_LA_VALEUR enfant1
22     TANT_QUE (nouvelenfant ==enfant1) FAIRE
23       DEBUT_TANT_QUE
24       nouvelenfant PREND_LA_VALEUR floor(2*random())
25       SI (nouvelenfant==0) ALORS
26         DEBUT_SI
27         AFFICHER "l'enfant suivant est un garçon"
28         FIN_SI
29         SINON
30           DEBUT_SINON
31           AFFICHER "l'enfant suivant est une fille"
32           FIN_SINON
33       NBENFANTS PREND_LA_VALEUR NBENFANTS + 1
34       FIN_TANT_QUE
35     AFFICHER "Stop !! !Plus de naissances !!!"
36     FIN_POUR
37   AFFICHER "Le nombre total d'enfants est "
38   AFFICHER NBENFANTS
39   MOYENNE PREND_LA_VALEUR NBENFANTS/100
40   AFFICHER "La moyenne du nombre d'enfants est :"
41   AFFICHER MOYENNE
42 FIN_ALGORITHME
```

3.3.22 Codage en Scilab

3.3.23 Codage en Python

Suites - Terminale

Soit la suite (u_n) définie sur $\mathbb{N}$ par :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = u_n - ln(u_n) \end{cases}$$

1. Ecrire un algorithme qui permettra à un utilisateur d'entrer au clavier un entier naturel n puis après traitement affichera le terme u_n
2. Ecrire un algorithme qui permettra à un utilisateur d'entrer au clavier un entier naturel n, d'entrer lui-même une valeur strictement positive pour u_0 puis après traitement affichera tous les termes de u_0 à u_n
3. Ecrire un algorithme qui permettra à un utilisateur d'entrer au clavier un entier naturel n, d'entrer lui-même une valeur strictement positive pour u_0 puis après traitement affichera la somme partielle $\sum_{k=0}^{n} u_k$

Algorithme 1 :

```
variables
   N : entier;
   U : réel;
debut algorithme
    (* entree des donnees *)
    lire(N);

    (* traitement *)
    si N = 0 alors debut
                            U := 1;
                fin
               sinon    debut
                            U := 1;
                            pour I variant de 1 à N
                              faire debut
                                        U := U - ln(U);
                                    fin;
                         fin;
    (* affichage des resultats *)
    afficher(U);
fin algorithme.
```

Algorithme 2

```
debut algorithme
    variables
     N : entier;
     U : réel;
    (* entree des donnees *)
    lire(N);
    lire(U) ;
    (* traitement et affichage des résultats *)
    si N = 0 alors debut
                            afficher(U);
                  fin
                sinon    debut
                            afficher(U);
                            pour I variant de 1 à N
                              faire debut
                                        U := U - ln(U);
                                        afficher(U);
                                   fin;
                           fin;
fin algorithme.
```

Algorithme 3

```
debut algorithme
    variables
     N : entier;
     U , S : réel;
    (* entree des donnees *)
    lire(N);
    lire(U) ;
    (* traitement et affichage des résultats *)
    S := U;
    si N = 0 alors debut
                            S := U;
                   fin
                sinon    debut
                            pour I variant de 1 à N
                              faire debut
                                        U := U - ln(U);
                                        S := S + U;
                                   fin;
                           fin;
       (* affichage des résultats *)
       afficher(S);
fin algorithme.
```

3.3.24 Codage en Scilab

3.3.25 Codage en Python

Chapitre 4

Procédures et Fonctions

4.1 Définition

Pour mieux déterminer des algorithmes, on utilise souvent la méthode "diviser pour régner".
On décompose le programme en une série de sous-programmes.
Ces sous-programmes qui sont :

- soient des procédures
- soient des fonctions

sont eux-mêmes structurés comme des programmes avec :

- une en-tête
- une partie déclarations qui peut contenir elle-même des procédures ou des fonctions.
  Les variables de la procédure s'appellent des **variables locales** à la procédure
- un corps débutant par begin et se terminant par end ;

On rend active une procédure ou une fonction par un appel de procédure ou un appel de fonction.

4.2 Sous-programmes procédures

> **Une procédure est une nouvelle instruction que l'on ajoute au programme.**
> **Une procédure modifie donc l'état de la machine.**

On distingue :

4.2.1 Les procédures sans paramètres

Exemple 1

```
procedure ATTENTE ;
                    begin
                            repeat
                            until keypressed;
                    end;
```

Exemple 2

```
procedure GAGNE;
                  begin
                           writeln('Vous avez gagné !!!');
                  end;
```

4.2.2 Les procédures avec paramètres

Dans le programme appelant, l'appel de la procédure comprend une liste de paramètres effectifs ou réels qui seront nécessaires à la procédure.
Dans l'en-tête de la procédure figure une liste de **paramètres formels ou abstraits.**

Exemple 3

```
program TOTO;
             var A , M : integer ;
             procedure Truc(B : integer);
                  begin
                         If B < M then B := M;
                  end;

             begin (* du programme principal TOTO *)
                    write('Entrez au clavier un entier A ='); readln(A);
                    write('Entrez au clavier un entier M =');readln(M);
                    Truc(A);
                    writeln('A =',A);
              end.(* du programme principal TOTO*)
```

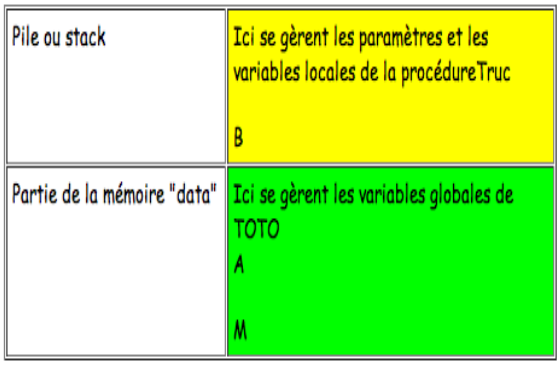
Dans l'écriture : Truc(A) , A est un paramètre réel ou effectif.
Dans l'écriture : procedure Truc(B :integer), B est un paramètre abstrait ou formel.

4.3 2 types de passage de paramètres

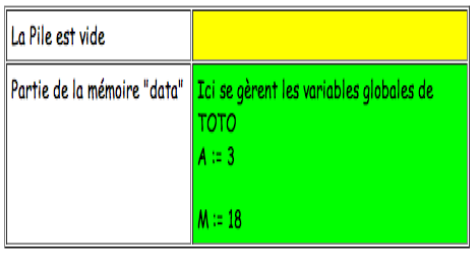
4.3.1 par valeur

```
program TOTO;
            var A , M : integer ;
            procedure Truc(B : integer);
                 begin
                        If B < M then B := M;
                 end;

            begin (* du programme principal TOTO *)
                   write('Entrez au clavier un entier A ='); readln(A);
                   write('Entrez au clavier un entier M =');readln(M);
                   Truc(A);
                   writeln('A =',A);
             end.(* du programme principal TOTO*)
```

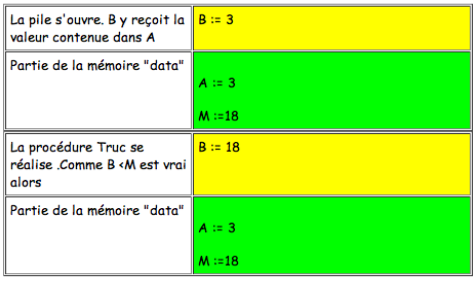
Les variables locales aux procédures et aux fonctions sont gérées dans une partie de la mémoire vive qu'on appelle la pile ou stack qui se vide à la fin de la procédure. Le paramètre formel de



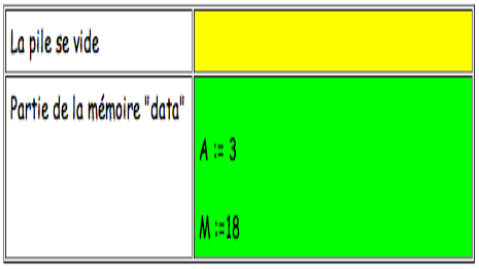
la procédure (ici B) va recevoir **la valeur du paramètre effectif** au moment de l'appel de la procédure.
Ce paramètre formel devient une variable locale à la procédure.
La valeur du paramètre effectif (ici A) n'est en aucun cas modifié par l'exécution de la procédure.
Supposons que l'utilisateur tape au clavier dans l'ordre 3 puis 18.



Appel Truc(A)



Fin de la procédure Truc Retour à l'instruction suivant l'appel Truc(A) ;

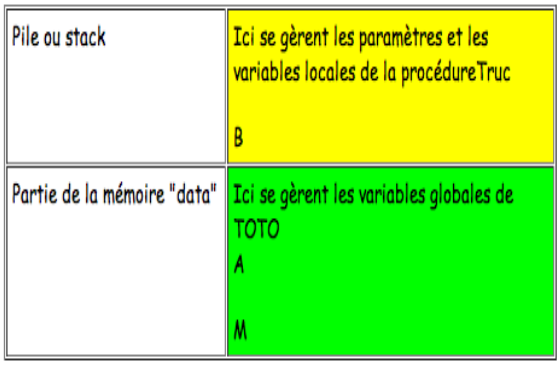


Writeln('A=',A)
A l'écran, il est affiché 3 ! ! !

4.3.2 par adresse ou par référence ou par variable

```
program TOTO;
             var A , M : integer ;
             procedure Truc(var B : integer);
                  begin
                         If B < M then B := M;
                  end;

             begin (* du programme principal TOTO *)
                    write('Entrez au clavier un entier A ='); readln(A);
                    write('Entrez au clavier un entier M =');readln(M);
                    Truc(A);
                    writeln('A =',A);
              end.(* du programme principal TOTO*)
```

Dans l'écriture : Truc(A) , A est un paramètre réel ou effectif.
Dans l'écriture : procedure Truc(var B :integer), B est un paramètre abstrait ou formel **passant cette fois-ci par adresse ou par référence ou par variable**
Au moment de l'appel de la procédure, A était le nom d'une case-mémoire située à une adresse



hexadécimale précise par exemple $F2CFA9F$.
Alors ce paramètre formel B devient le nouveau nom de cette case mémoire.
La case-mémoire située à l'adresse $F2CFA9F$ a donc 2 noms ou deux étiquettes A et B.
A et B ont donc la même référence ou la même adresse.
Ce sont 2 noms différents d'une même variable.
Donc tout changement affectant B va donc affecter A!!!.
La valeur du paramètre effectif A est modifiée par l'exécution de la procédure.
Supposons par exemple que l'utilisateur tape au clavier dans l'ordre 3 puis 18

La Pile est vide	
Partie de la mémoire "data"	Ici se gèrent les variables globales de TOTO A := 3 M := 18

Appel Truc(A)

La pile s'ouvre. B y reçoit la valeur contenue dans A	B := 3
Partie de la mémoire "data"	A := 3 M :=18
La procédure Truc se réalise .Comme B <M est vrai alors	B := 18
Partie de la mémoire "data" Le contenu de A est donc modifiée	A := 18 M :=18

Fin de la procédure Truc
Retour à l'instruction suivant l'appel Truc(A) ;

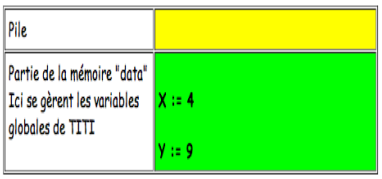
La pile se vide	
Partie de la mémoire "data"	A := 18 M :=18

Writeln('A=',A)
A l'écran, il est affiché 18 ! ! !

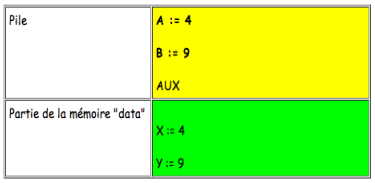
4.3.3 Un autre exemple de passage par adresse

```
program TITI;
           var X , Y : integer;

           procedure ECHANGE(var A : integer;var B :integer);
              var AUX : integer;
                    begin
                       AUX := A;
                       A := B;
                       B := AUX;
                     end;

      begin (* du programme principal TITI *)
                        write('Entrez au clavier un entier X ='); readln(X);
                         write('Entrez au clavier un entier Y =');readln(Y);
                         ECHANGE(X,Y);
                         writeln('X ='),X);
                         write('Y =', Y);
       end.(* du programme principal TITI *)
```

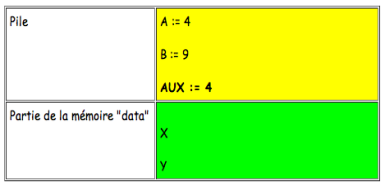
Supposons qu'un utilisateur tape au clavier dans l'ordre 4 puis 9

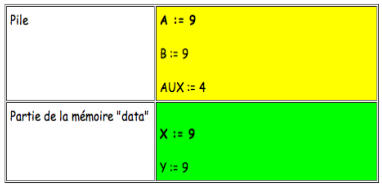


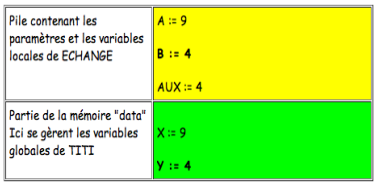
Appel de ECHANGE



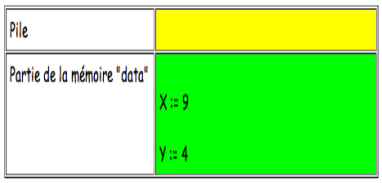
Exécution de ECHANGE







Fin de la procédure ECHANGE



A l'affichage de X , c'est 9 et pour Y c'est 4 ! ! !
On a ainsi permuté le contenu des deux cases mémoires X et Y

4.4 Sous-programmes fonctions

> **Une fonction ramène un résultat.**
> L'en-tête d'une fonction ressemble à celle d'une procédure . Il diffère sur un point :
> le mot réservé est function suivi d'abord entre parenthèses de la liste des paramètres formels puis du type du résultat ramené.
> Pour une fonction, le passage de paramètres a lieu par valeur ou par adresse comme pour les procédures.

4.4.1 Fonction puissance

```
function puissance(A : word ; B : real) : real;
            begin
                    puissance := exp(B * ln(A));
            end;
```

le type du résultat ramené peut être de tout type scalaire prédéfini ou non, de type string ou de type pointeur. C'est la dernière affectation de l'identificateur de la fonction qui détermine le résultat de la fonction.

4.4.2 Fonction maximum trois entiers

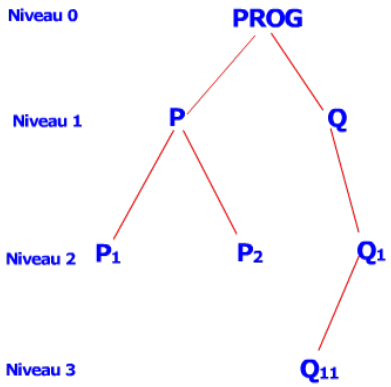
```
Program TATA ;
    Var X, Y , Z : integer;
    function MAX3(var A : integer; var B : integer ; var C : integer): integer;

               function MAX2( var D : integer; var E : integer);
                     begin (* de MAX2 *)
                              If D < E then MAX2 := E
                                           Else MAX2 := D
                      end(* de MAX2 *)

              begin (* de MAX3 *)
                      MAX3 := MAX2(A, MAX2(B,C));
              end; (* de MAX3 *)

    begin (* de TATA *)
           write('Tapez au clavier X ='); readln(X);
           write('Tapez au clavier Y ='); readln(Y);
           write('Tapez au clavier Z ='); readln(Z);
           write(Le plus grand des 3 entiers est =', MAX3(X,Y,Z));
    end(* de TATA *)
```

4.5 Niveaux de sous-programmes

L'ensemble des déclarations des sous-programmes forme une hiérarchie de sous-programmes. Dans cette hiérarchie, chaque sous-programme a un niveau particulier.

```
program PROG;
   procedure P;
     procedure P1;
       begin (* de P1 *)
        ............
       end (* de P1 *)
     procedure P2
       begin (* de P2 *)
          ............
       end (* de P2 *)
   begin (* de P *)
     ............
   end (* de P *)
  procedure Q;
     procedure Q1;
        procedure Q11;
          begin (* de Q11 *)
                ............
          end (* de Q11 *)
       begin (* de Q1 *)
        ............
       end (* de Q1 *)
    begin (* de Q *)
     ............
    end (* de Q *)

 begin (* de P *)
     ............
 end (* de P *)
```

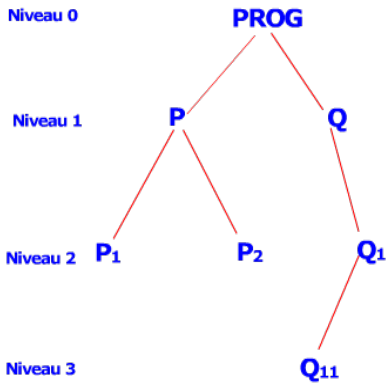
Cette hiérarchie peut se représenter par une arborescence :



4.6 Variables globales et variables locales

1. Tous les objets(const, type, var, procedure, function) qui sont déclarés dans un sous-programme sont dits locaux à ce sous-programme.

2. Les objets locaux à une procédure sont inaccessibles :
   - par le programme appelant
   - par les sous-programmes déclarés dans le même bloc
   - par les sous-programmes englobant
3. Lorsque dans une procédure, on déclare des variables locales, ces dernières sont créées lors de l'appel de la procédure.
   Elles sont restituées lors de l'appel de la procédure.
4. Un objet déclaré dans un sous-programme S à un niveau I est inaccessible aux sous-programmes de niveau $\leq I$ et dans les programmes de niveau $> I$ qui ne sont pas englobés par S.
5. Par exemple, les objets locaux à Q_1 sont inaccessibles à P_1, P_2, Q mais sont accessibles à Q_{11}.
6. Les identificateurs connus dans un bloc sont ceux qui ont été déclarés localement dans un bloc ou dans les blocs englobant de niveau inférieur.
7. Dans le cas où il y a homonymie entre des identificateurs, l'identificateur fait référence à celui qui est déclaré dans le bloc le plus proche.



4.7 Exercices

4.7.1 Exercice

```
program A;
     Var X , Y , Z : ....;
     procedure B;
         Var X : ....;
         procedure C;
               Var Y :....;
               procedure D;
                    var X, Y : ...;
                    begin
                      .........
                    end;
                begin
                      ............
                 end;
           procedure E;
                Var P : ...;
                 begin
                    .........
                 end;
           begin
             .........
           end;
begin
  ..........
end.
```

1. Représenter l' arborescence des procédures
2. Compléter le tableau suivant avec des OUI ou des NON :

Objets	accessibles par A	accessibles par B	accessibles par C	accessibles par D	accessibles par E
X de D					
P de E					
X de A					
X de B					
Y de D					
Y de A					
Y de C					
Z					

4.7.2 Exercice

Soit le programme suivant :

```
program TUTU ;
     Var A, B, C : integer;
     procedure SUB( D : integer; var E : integer; C : integer);
           var A : integer;
           begin (* de SUB *)
                  A := C + 1 ;
                  E := A + C ;
                  C := C * 2 ;
           end; (* de SUB *)
 begin (* de TUTU *)
      write('A =');readln(A);
      write('B =');readln(B);
      write('C =');readln(C);
      SUB(B,C,A);
      writeln('A =',A);
      writeln('B =',B);
      writeln('C =',C);
end.
```

Si un utilisateur , lors de l'exécution de ce programme, entre au clavier les valeurs 4, 5 et 6, indiquer dans l'ordre les 3 valeurs affichées après traitement à l'écran.

4.7.3 Exercice

Indiquer les résultats affichés par ce programme. Justifier.

```
program P;
     var t : real;
     function f1(X : real) : real;
          begin
               f1:= X * X;
               X := X + 1;
          end;
     function f2(var X: real): real;
          begin
               f2:= X * X;
               X := X + 1;
           end;
     begin
           t:=1 ;
           writeln(f1(t));
           writeln(f2(t));
           writeln(f2(t));
           writeln(f1(t));
           writeln(t);
    end.
```

Chapitre 5

La Récursivité

5.1 Définition

Un langage est dit récursif lorsque dans le corps d'une procédure (resp. d'une fonction) on peut faire appel à cette procédure (resp. cette fonction) ou y faire référence.
La récursivité peut être :

- directe : A appelle A)
- indirecte ou croisée : A appelle B et B appelle A.

Voisine de la récurrence mathématique ,la récursivité est plus qu'une technique de programmation ,c'est une **METHODE DE REFLEXION** qui permet de régler des problèmes difficiles en particulier en Intelligence Artificielle , en Systèmes Experts , en Analyse syntaxique, en Calcul Symbolique par ordinateur.

LA RECURSIVITE DIVISE POUR REGNER

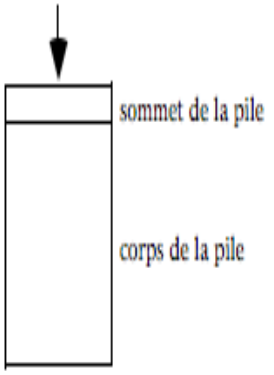
Un problème complexe est transformé en plusieurs sous-problèmes similaires plus simples et ainsi de suite ... jusqu'au moment où on est ramené à des cas suffisamment simples pour être traités directement.
On la rencontre dans de nombreux langages de programmation : PASCAL ,LSE , LOGO ,LISP,PROLOG,CAML,...

Le logo de la boîte du fromage '' Vache qui rit '' est un modèle d'exemple de logo récursif.

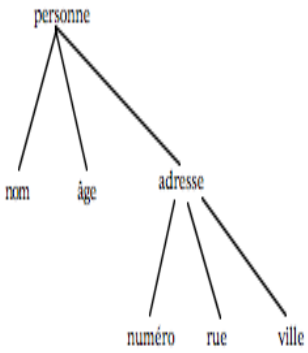
Beaucoup d'objets en Informatique sont définis récursivement :

- Une **LISTE** = est une structure de données qui a la forme suivante :
  [1er terme ou tête de liste : le reste ou la queue de la liste qui est elle-même une liste]
  Exemple : La liste [1 2 3] = [1 : [2 3]] où [2 3] = [2 : [3]] avec [3 : [ ]] où [ ] est la liste vide.

- Une **PILE** est une structure de données de type LIFO (Last In ,First Out) formée d'un sommet de pile et d'un corps de pile qui est lui-même une pile.



- Un **ARBRE** de type T est une structure de données formée :
  — d'une donnée de type T appellée RACINE
  — d'un ensemble fini d'arbres de type T ,appelés SOUS-ARBRES de l'arbre. Cet ensemble peut être vide.



Lorsqu'on manie la récursivité , il faut utiliser 2 idées de base :

1. **1ère idée :**
   éviter que la procédure ne tourne indéfiniment. La profondeur du processus doit être mesurable (< 1000) et connue sinon il faut itérer car l'on risque de saturer la mémoire. En Turbo-Pascal ,ne pas dépasser une profondeur de 300.
2. **2ème idée :** L'écriture d'une procédure récursive est l'écriture d'une définition avec un ordre précis :

```
en-tête
traitement des cas particuliers
une définition qui nous rapproche de la solution
fin
```

Attention lorsqu'on raisonne en récursivité ,il faut comme pour l'itération :

- d'abord traiter le cas général en cherchant une démarche qui nous rapproche de la solution
- ensuite initialiser

5.2 Exemples

5.2.1 la fonction puissance

en récursif

```
function PUISSANCE(x : word ; n : word):word;
begin
     if  n = 0 then PUISSANCE := 1
               else  PUISSANCE := x *  PUISSANCE(x,n-1)
end;
```

en utilisant la définition mathématique

```
function PUISSANCE(x : real ; n : real) : real;
begin
      PUISSANCE := exp( n *  ln(x))
end;
```

en itératif

```
function PUISSANCE(X : word ; N : word):word;
var P ,I : word;
begin
if N = 0
        then PUISSANCE := 1
        else begin
                 P := 1;
                 for I := 1 to N do P := P * X ;
                 PUISSANCE := P
             end ;
end;
```

5.2.2 la fonction factorielle

en itératif - Turbo Pascal

```
function FACTORIELLE(N : word) : word;
var A : real;
      I : word ;
begin
if (( N = 0) or (N = 1))
        then FACTORIELLE := 1
        else begin
                 A := 1;
                 for I := 2 to N do A := A * I ;
                 FACTORIELLE := A
             end ;
end;
```

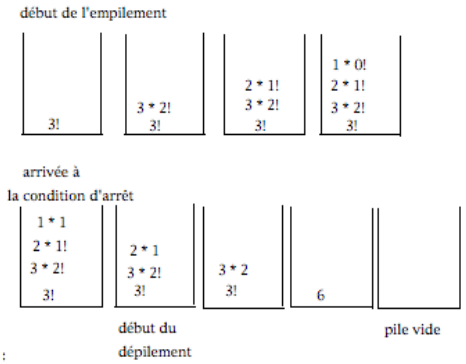
en récursif - Turbo Pascal

```
function FACTORIELLE(N : word) : word;
begin
  if N = 0 then FACTORIELLE := 1
           else FACTORIELLE := N *  FACTORIELLE(N - 1)
end;
```

en récursif - Maple

```
fact :=proc(n :: nonnegint);
  if n = 0 then return(1)
           else return(n * fact(n - 1))
           fi;
          end;
```

Illustrons ce qui se passe pour n = 3 en utilisant la pile des variables locales



5.2.3 Fibonacci

en itératif - Maple

En utilisant la structure de tableau

```
fibotable :=proc(n :: nonnegint);
           local k, f;
           f[0]:=0;f[1]:=1;
             for k from 2 to n do
                                  f[k] :=f[k-1] + f[k-2];
                             od;
           return(f[n]);
          end;
```

en itératif - Maple

On simplifie ici la complexité spatiale : au lieu d'utiliser k plus les n cases du tableau f on utilise uniquement 4 cases -mémoire : k, a, b, c

```
fibo :=proc(n :: nonnegint);
           local a, b, c , k ;
              if n = 0 or n = 1
                 then return(n)
                 else
                  a:=0 ;
                  b :=1;
                  for k from 2 to n
                   do
                     c:= a + b;
                     a:= b;
                     b:= c ;
                   od;
                   return(c)
           fi;
          end;
```

En utilisant les formules de BINET - Maple

```
fbinet :=proc(n :: nonnegint);
              local phi,psi,sols;
                sols:=solve(x*x - x - 1,x);
                phi :=sols[1];psi:=sols[2];
                return(expand((phi^n - psi^n)/sqrt(5)))
          end;
```

en récursif - Maple

```
fiborec :=proc(n :: nonnegint);
               option remember;
            if n = 0
              then return(1)
               else return(fiborec(n - 1) + fiborec(n - 2))
            fi;
          end;
```

5.2.4 Algorithme d'Euclide pour le PGCD

Itératif

```
program PGCDITERATIF;
 Uses WinCrt;
 var X,Y : longint;
 procedure ECHANGE(var U :longint , var V : longint);
   var W : longint;
    begin
      W := U ; U := V ; V := W ;
    end;
 function PGCD1(A,B:longint) : longint;
   var RESTE : longint;
   begin
        repeat
          RESTE := A mod B;
          A := B ;
          B := RESTE;
       until RESTE = 0;
       PGCD1 := A;
  end;

  begin
    repeat
      write('Saisissez au clavier un entier X = ');
      readln(X)
   until ((X >0) and (X = trunc(X));
   repeat
      write('Saisissez au clavier un entier Y = ');
      readln(Y)
   until ((Y >0) and (Y = trunc(Y));
   if X < Y then ECHANGE(X,Y);
   writeln('Le pgcd de ', X , ' et de ', Y , ' est  : '),PGCD(X,Y));
  end.
```

On peut toujours en itératif utiliser l'algorithme des différences successives dans la fonction PGCD :

```
 function PGCD2(A,B:longint) : longint;
   var RESTE : longint;
   begin
          RESTE := A ;
          while RESTE >= B
          do begin
                 RESTE := RESTE - B;
               end;
           PGCD2 := RESTE;
  end;
```

Récursif

```
program PGCDITERATIF;
 Uses WinCrt;
 var X,Y : longint;
 procedure ECHANGE(var U :longint , var V : longint);
   var W : longint;
    begin
      W := U ; U := V ; V := W ;
    end;
 function PGCDR(A,B:longint) : longint;
   var RESTE : longint;
   begin
          RESTE := A mod B;
          if RESTE = 0 then PGCDR := A
                           else PGCDR :=PGCD(Y,RESTE);
 end;

 begin
   repeat
     write('Saisissez au clavier un entier X = ');
     readln(X)
  until ((X >0) and (X = trunc(X));
  repeat
     write('Saisissez au clavier un entier Y = ');
     readln(Y)
  until ((Y >0) and (Y = trunc(Y));
  if X < Y then ECHANGE(X,Y);
  writeln('Le pgcd de ', X , ' et de ', Y , ' est  : '),PGCDR(X,Y));
 end.
```

5.2.5 Anagrammes

2 mots A et B sont-ils des anagrammes ? On utilisera les fonctions suivantes :

- pos(caractère,chaîne de caractères) qui ramène la position du caractère dans la chaîne de caractères sinon 0.
- length(chaîne) qui ramène le nombre de caractères de la chaîne
- copy(c, p, n) qui ramène la sous chaîne extraite de la chaîne c démarrant à partir de la position de départ p et comportant n caractères.

```
function ANAGRAMME(A,B:string):boolean;
var K,L:0..255;
     begin
       L:= length(A);
       if length(B) <>  L
          then ANAGRAMME := false
          else if lenght(A) = 1
                 then ANAGRAMME := (A=B)
                 else begin
                   K := pos(copy(A,1,1),B);
                   if K= 0 then ANAGRAMME := false
                           else
                               ANAGRAMME:=ANAGRAMME(copy(A,2,L-1),copy(B,1,K-1)+copy(B,K+1,L-K))

               end
     end ;
```

5.2.6 Les Tours de Hanoï

Une légende qui remonte à la nuit des temps.

" Dans le fameux temple de Bénarès, sous le dôme marquant le cente du monde, trône un socle de bronze sur lequel sont fixées trois aiguilles de diamant, chacune d'elles haute d'une coudée et fine come la taille d'une guêpe. Sur une de ces aiguilles, à la Création, Dieu empila 64 *disques d'or pur, du plus grand au plus petit, le plus large reposant sur le socle de bronze. Il s'agit de la Tour de Brahma.*
Jour et nuit, inlassablement, les moines déplacent les disques d'une aiguille vers l'autre tout en respectant les immuables lois de Brahma qui obligent les bonzes à ne déplacer qu'un disque à la fois et à ne jamais le déposer sur un disque plus petit.
Quand les 64 *disques auront été déplacés de l'aiguille sur laquelle Dieu les déposa à la Création vers une des autres aiguilles, la tour, le temple et les brahmanes seront réduits en poussière, et le monde disparaîtra dans un grondement de tonnerre. "*
Le but de ce jeu(inventé par Edouard Lucas en 1883) est **de déplacer** n **rondelles** ($n \in \mathbb{N}^*$) **de la tige** 1 **à la tige** 3 **en un nombre minimum de coups.** Un coup consiste à déplacer une rondelle située au sommet d'une poule au sommet d'une autre pile.
Les règles du jeu sont les suivantes :

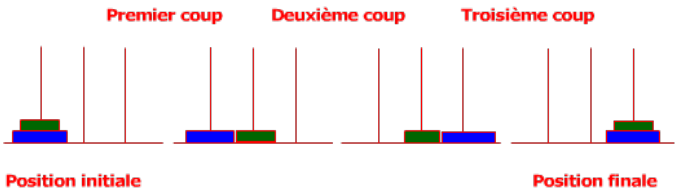
- On ne déplace qu'une seule rondelle à la fois
- Une rondelle ne doit jamais se retrouver au-dessus d'une rondelle plus petite.

Soit $n \in \mathbb{N}^*$. On appellera u_n le nombre minimal de coups permettant de déplacer n rondelles de la tige 1 à la tige 3

$n = 1$

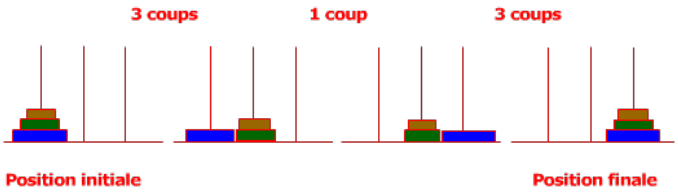
Le nombre minimal de coups permettant de déplacer 1 rondelle de la tige 1 à la tige 3 est $u_1 = 1$

$n = 2$



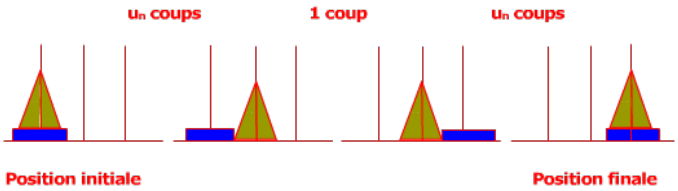
Le nombre minimal de coups permettant de déplacer 2 rondelles de la tige 1 à la tige 3 est $u_2 = 3$.

$n = 3$



Le nombre minimal de coups permettant de déplacer 3 rondelles de la tige 1 à la tige 3 est $u_3 = 7$

Cas général



Le procédé utilisé (dans le cas $n = 3$) permet de passer du cas de n rondelles au cas de $n + 1$ rondelles : il suffit de traiter un lot de n rondelles lorsqu'il y en a $n+1$ comme celui de 2 rondelles dans le cas où il y en avait 3.
Alors $u_{n+1} = 2u_n + 1 = f(u_n)$ où f est la fonction affine $x \mapsto 2x + 1$

Etude de la suite (u_n)

1. $(u_n)_{n\in\mathbb{N}^*}$ est une suite arithmético-géométrique car $u_{n+1} = au_n + b$ avec $a = 2$ et $b = 1$.
2. Alors le point fixe L tel que $f(L) = L$ est $L = \dfrac{b}{1-a} = -1$
3. Alors la suite $(v_n)_{n\in\mathbb{N}^*}$ définie par $v_n = u_n - L = u_n + 1$ est géométrique de raison $q = 2$ et de premier terme $v_1 = u_1 + 1 = 2$ car :
   $v_{n+1} = u_{n+1} + 1 = 2u_n + 1 + 1 = 2u_n + 2 = 2(u_n + 1) = 2v_n$
4. Donc $\forall n \in \mathbb{N}^*$ on a donc $v_n = q^{n-1}v_1 = 2^{n-1} \times 2 = 2^n$ d'où $\boxed{u_n = v_n - 1 = 2^n - 1}$
5. Par conséquent , $\boxed{u_{64} = 2^{64} - 1 = 18446744073709551616 \approx 1.844674407 \times 10^{19}}$
6. Sachant qu'une année comporte 365 jours de 24 heures , chaque heure durant $60'$ et une minute durant 60", en supposant qu'un déplacement dure une seconde, alors les 64 disques auront été déplacés en :
   $\boxed{\dfrac{18446744073709551616}{365 \times 24 \times 60 \times 60} \approx 5.849424174 \times 10^{11} \text{ années}}$.

Algorithme récursif en Turbo Pascal

On utilisera la remarque suivante, comme la somme des numéros des 3 tiges est $1+2+3=6$ donc si l'on appelle D le numéro d'une tige de départ, A le numéro de la tige d'arrivée et I le numéro de la tige intermédiaire alors $D+I+A=6$ donc $\boxed{I=6-D-A}$

```
program HANOI_EN RECURSIF ;
uses CRT ;
var N : integer ;
     procedure HANOYER(P,D,A:integer);
        procedure BOUGEDISQUE(X,U,V : integer);
         begin
            writeln('Le disque n° ',X,' va de la tige ',U, '  à la tige ',V)
         end;

         begin (* de HANOYER *)
            if P = 1 then BOUGEDISQUE(1,D,A)
                     else  begin
                               HANOYER(P-1 , D , 6 - D - A) ;
                               BOUGEDISQUE(P,D,A);
                               HANOYER(P - 1 , 6 - D - A , A);
                           end
        end ; (* de HANOYER *)

 begin (* programme principal *)
     clrscr;
     writeln(' Nous allons jouer au jeu des Tours de Hanoï ');
     writeln(' La tige de départ est la tige 1 ');
     writeln(' La tige d''arrivée  est la tige 3 ');
     writeln(' La tige intermédiaire est la tige 2 ');
     writeln;
     write(' Tapez au clavier le nombre de disques N = ');
     readln(N) ;
     HANOYER(N,1,3);
     writeln;
     writeln(' J''ai terminé de jouer ! ')
end.
```

Algorithme en Maple :

5.2.7 PGCD des entiers A et B

```
function PGCD ( A,B :integer):integer;
var
begin
if ( (A = 1 ) or ( B = 1 ) )
then  PGCD := 1
else if A = B  then PGCD := A
          else if  A < B  then PGCD := PGCD ( A , B - A )
         else  PCGD := PGCD ( B , A - B )
end;
```

5.3 Comparaison entre récursivité et itération

Ce sont 2 univers différents :

1. **En récursivité :**
   - On a l'existence d'une solution.
   - On se borne à fournir une définition qui va nous rapprocher de la solution.
   - Par exemple , inverser récursivement une chaine de caractères
     — inverser 'TRI' c'est inverser 'RI' et concaténer avec 'T'
     — inverser 'RI' c'est inverser 'I' et concaténer avec 'R'
     — inverser 'I' c'est 'I'

```
function  INVERSE(CHAINE:string):string;
var L : 0..255 ;
begin
     L := length(CHAINE)
     if (( L = 0 ) or ( L = 1 ))
                   then INVERSE := CHAINE
                   else INVERSE := INVERSE(copy(CHAINE,2,L-1))+copy(CHAINE,1,1))
end;
```

2. **En itération :**
   - On en sait plus . Elle demande bien sûr plus d'efforts mais la démarche est moins abstraite. On la perçoit plus vite.
   - Seul problème psychologique dans la démarche itérative : "Supposons que l'on ait fait une partie du programme, comment continuer "
   - On peut la faire tourner à la main.

- Inversons une chaîne de caractères itérativement :
  On prend la lettre en-tête ,on la pose et on recommence avec ce qui reste

```
function INVERSE (CHAINE : string) : string;
var D :string ;
    L: 0..255;
begin
     D := '';
     while CHAINE <>  '' do begin
                              D := copy(CHAINE,1,1);
                              L:= length(CHAINE);
                              CHAINE := copy(CHAINE , 2 , L-1)
                           end;
     INVERSE := D
end;
```

Chapitre 6

Preuve et Complexité algorihmique

"Le contrôle de la complexité est l'essence de la programmation informatique"
Brian W. Kernighan

6.1 Qualités d'un programme

Un programme doit être :

1. **juste :**
   Il doit bien répondre au problème posé. La preuve d'un programme est complexe et ce n'est qu'à la fin des années 1970 que l'on s'est posé le problème de la justesse d'un programme.
2. **efficace :**
   Il doit se terminer en un minimum de temps , en utilisant le minimum de place en mémoire

6.2 Algorithmes de calcul d'un coefficient binomial

Un calcul numérique doit être conduit de manière à ce que les résultats intermédiaires aient des valeurs sympathiques c'est-à-dire pas trop grandes et ,si possible sans erreur d'arrondi.
Ceci est encore vrai s'il s'agit d'un calcul programmé où l'on va essayer de minimiser le nombre d'additions, de soustractions, de multiplications et de divisions.
Voici trois méthodes systématiques (programmables) de calcul de $\binom{n}{p}$

1. **Méthode 1** : $a = n!$; $b = p!$; $c = (n-p)!$; $d = b \times c$; $\binom{n}{p} = \frac{a}{d}$
2. **Méthode 2** : $a_1 = \frac{n}{p}$; $a_2 = \frac{a_1 \times (n-1)}{p-1}$; $a_3 = \frac{a_2 \times (n-2)}{p-2}$; $\cdots$

jusqu'à obtenir $\binom{n}{p} = a_p = \dfrac{a_{p-1} \times (n-p+1)}{p}$

3. **Méthode 3** : $a_1 = \dfrac{n}{1}$; $a_2 = \dfrac{a_1 \times (n-1)}{2}$; $a_3 = \dfrac{a_2 \times (n-2)}{3}$; $\cdots$
   jusqu'à obtenir $\binom{n}{p} = a_p = \dfrac{a_{p-1} \times (n-p+1)}{p}$

1. Combien y-a-t-il de multiplications dans le calcul itératif de $n!$?
2. Calculer $\binom{9}{4}$ par les 3 méthodes. Pour chacune des 3 méthodes vous indiquerez le nombre total d'additions éventuelles, de soustractions éventuelles, de multiplications éventuelles , de divisions éventuelles.Expliquer pourquoi la méthode 3 est meilleure que les deux autres
3. Voici deux algorithmes.
   Indiquez pour chaque algorithme la méthode 1, 2 ou 3 qu'il représente.
   (a) **Début Algorithme 1**
   $A := 1$;$B := 0$; $P := p$;$N := n$;
   Répéter
   $A := A \times \dfrac{N-B}{B+1}$;
   $B := B + 1$;
   jusqu'à $B \geq P$
   Afficher A
   fin.
   (b) **Début Algorithme 2**
   $A := 1$;$P := p$; $N := n$;
   Pour $B := 1$ à P faire
   début
   $A := A \times \dfrac{N-B+1}{B}$;
   fin ;
   Afficher A
   fin.

Corrigé

1. Il y a exactement (n-1) multiplications dans le calcul itératif de $n!$ car $n! = 1 \times 2 \times 3 \times 4 \times \cdots \times (n-1) \times n$
2. Calcul de $\binom{9}{4}$ par les 3 méthodes.
   (a) **Méthode 1** : $a = n!$; $b = p!$; $c = (n-p)!$; $d = b \times c$; $\binom{n}{p} = \dfrac{a}{d}$
   - $a = n! = 9! = 1 \times 2 \times 3 \times 4 \times \cdots \times 8 \times 9$ donc 8 multiplications, 0 divisions et 0 soustractions
   - $b = p! = 4! = 1 \times 2 \times 3 \times 4$ donc 3 multiplications, 0 divisions et 0 soustractions
   - $c = (n-p)! = 5! = 1 \times 2 \times 3 \times 4 \times 5$ donc 4 multiplications, 0 divisions et 1 soustractions
   - $d = b \times c$ donc 1 multiplications, 0 divisions et 0 soustractions
   - $\dfrac{9!}{4!5!}$ 0 multiplications, 1 divisions et 0 soustractions

En tout le calcul de $\binom{9}{4} = \frac{9!}{4!(9-4)!}$ nécessite 18 opérations : 16 multiplications, 1 divisions et 1 soustractions

(b) **Méthode 2** : $a_1 = \frac{n}{p}$; $a_2 = \frac{a_1 \times (n-1)}{p-1}$; $a_3 = \frac{a_2 \times (n-2)}{p-2}$; $\cdots$
jusqu'à obtenir $\binom{n}{p} = a_p = \frac{a_{p-1} \times (n-p+1)}{p}$

- $a_1 = \frac{n}{p} = \frac{9}{4}$ donc 0 multiplications, 1 divisions et 0 soustractions
- $a_2 = \frac{a_1 \times (n-1)}{p-1} = \frac{9}{4} \times \frac{8}{3}$ donc 1 multiplications, 1 divisions et 2 soustractions
- $a_3 = \frac{a_2 \times (n-2)}{p-2} = \frac{9}{4} \times \frac{8}{3} \times \frac{7}{2}$ donc 1 multiplications, 1 divisions et 2 soustractions
- $a_4 = \frac{a_3 \times (n-3)}{p-3} = \frac{9}{4} \times \frac{8}{3} \times \frac{7}{2} \times \frac{6}{1}$ donc 1 multiplications, 1 divisions et 2 soustractions

En tout le calcul de $\binom{9}{4} = \frac{9!}{4!(9-4)!}$ nécessite 13 opérations : 3 multiplications, 4 divisions et 6 soustractions

(c) **Méthode 3** : $a_1 = \frac{n}{1}$; $a_2 = \frac{a_1 \times (n-1)}{2}$; $a_3 = \frac{a_2 \times (n-2)}{3}$; $\cdots$
jusqu'à obtenir $\binom{n}{p} = a_p = \frac{a_{p-1} \times (n-p+1)}{p}$

- $a_1 = \frac{n}{1} = \frac{9}{1}$ donc 0 multiplications, 1 divisions et 0 soustractions
- $a_2 = \frac{a_1 \times (n-1)}{2} = \frac{9}{1} \times \frac{8}{2}$ donc 1 multiplications, 1 divisions et 1 soustractions
- $a_3 = \frac{a_2 \times (n-2)}{3} = \frac{9}{1} \times \frac{8}{2} \times \frac{7}{3}$ donc 1 multiplications, 1 divisions et 1 soustractions
- $a_4 = \frac{a_3 \times (n-3)}{4} = \frac{9}{1} \times \frac{8}{2} \times \frac{7}{3} \times \frac{6}{4}$ donc 1 multiplications, 1 divisions et 1 soustractions

En tout le calcul de $\binom{9}{4} = \frac{9!}{4!(9-4)!}$ nécessite 10 opérations : 3 multiplications, 4 divisions et 3 soustractions

La méthode 3 est donc meilleure que les deux autres

3. Les deux algorithmes sont deux versions différentes de la méthode 3 .

(a) **Début Algorithme 1**
$A := 1$;$B := 0$; $P := p$;$N := n$;
Répéter
$A := A \times \frac{N-B}{B+1}$;
$B := B + 1$;
jusqu'à $B \geq P$
Afficher A
fin.

(b) **Début Algorithme 2**
$A := 1$;$P := p$; $N := n$;
Pour $B := 1$ à P faire
début

$A := A \times \dfrac{N - B + 1}{B}$;
fin ;
Afficher A
fin.

6.3 Exercice : Longueur de l'algorithme d'Euclide

Le calcul du pgcd $a \wedge b$ de 2 entiers a et b $(a > b > 0)$ par l'algorithme d'Euclide se fait par divisions successives.
Exemple : $a = 44$; $b = 18$

$$44 = 2 \times 18 + 8$$

$$18 = 2 \times 8 + 2$$

$$8 = 4 \times 2 + 0$$

$$44 \wedge 18 = 2$$

Si on désigne par $l(a, b)$ la longueur de l'algorithme, c'est-à-dire le nombre de divisions nécessaires pour aboutir au résultat, nous avons ici :

$$l(44, 18) = 3$$

L'objet de cet exercice est de majorer $l(a, b)$. Pour cela, on note

$$r(1), r(2), r(3), \cdots, r(n)$$

les restes des n divisions successives . On a donc $l(a, b) = n$ et $r(n) = 0$.
On note par convention $a = r(-1)$ et $b = r(0)$.

1. Montrer que l'algorithme d'Euclide se termine.
2. Montrer que pour $1 \leq k \leq n - 1$, on a $r(k - 2) \geq r(k - 1) + r(k)$
3. Soit $(F(n))$ la suite de Fibonacci définie par :

   $$F(0) = F(1) = 1$$

   $$F(n) = F(n - 2) + F(n - 1) \textit{ pour } n \geq 2$$

   Montrer qu'en posant $\alpha = \dfrac{1 + \sqrt{5}}{2}$, on a $\alpha^{n-1} \leq F(n)$
4. Montrer que $\forall k \in \mathbb{N}$, on a

   $$r(n - k - 1) \geq F(k)$$

   et en déduire que n vérifie une majoration de la forme :

   $$n \leq A \ln(a) + B$$

5. Vérifier cette majoration sur l'exemple $a = 44$ et $b = 18$. On pourra prendre $B = 1$ et $A < 2, 1$

6.3.1 Corrigé

Division euclidienne

Si a est un entier naturel et b un entier naturel non nul alors il existe un couple unique (q, r) d'entiers naturels tels que $a = bq + r$ avec $0 \leq r < b$.
q s'appelle le quotient de la division euclidienne de a par b
r s'appelle le reste de la division euclidienne de a par b

PGCD

Comme $a = bq + r$ alors tout diviseur commun à a et b est un diviseur commun à a , à b et à bq donc est un diviseur commun à b et à $a - bq$ donc est un diviseur commun à b et à r.
Réciproquement, tout diviseur commun à b et à r est un diviseur commun à b, à bq et à r donc est un diviseur commun à b et à $bq + r$ c'est-à-dire un diviseur commun à a et à b
Théorème : l'ensemble des diviseurs communs à a et à b est l'ensemble des diviseurs communs à b et à r.

Algorithme d'Euclide de recherche du PGCD

1. Le calcul du pgcd $a \wedge b$ de 2 entiers a et b $(a > b > 0)$ par l'algorithme d'Euclide se fait par divisions successives.
   Exemple : $a = 44\,;\, b = 18$
   $$44 = 2 \times 18 + 8$$
   $$18 = 2 \times 8 + 2$$
   $$8 = 4 \times 2 + 0$$
   donc le $pgcd(44; 22) = 44 \wedge 18 = 2 =$ le dernier reste non nul $= 2$.
   Si on désigne par $l(a, b)$ la longueur de l'algorithme, c'est-à-dire le nombre de divisions nécessaires pour aboutir au résultat, nous avons ici :
   $$l(44, 18) = 3$$
2. Posons $n = L(a; b)$. On note $r(1), r(2), \cdots, r(n)$ la suite des restes des n divisions successives.
   Donc $r(n) = 0$. Par convention, on pose $r(-1) = a$ et $r(0) = b$.
3. L'algorithme d'Euclide se termine car :
   $$\begin{cases} a = r(-1) = q(0)\ b + r(1) = q(0)r(0) + r(1)\ avec\ 0 \leq r(1) < r(0) \\ b = r(0) = q(1)r(1) + r(2)\ avec\ 0 \leq r(2) < r(1) \\ r(1) = q(2)r(2) + r(3)\ avec\ 0 \leq r(3) < r(2) \\ \cdots \\ r(n-3) = q(n-2)r(n-2) + r(n-1))\ avec\ 0 \leq r(n-1) < r(n-2) \\ r(n-2) = q(n-1)r(n-1) + r(n)\ avec\ 0 = r(n) < r(n-1) \end{cases}$$
   La suite des restes $r(1), r(2), \ldots\ldots\ldots, r(n)$ est une suite strictement décroissante d'entiers naturels donc au bout d'un nombre n fini d'opérations on aura $r(n) = 0$.
   Par conséquent **l'algorithme d'Euclide se termine**. **De plus, cet algorithme est valide** car le dernier reste non nul obtenu $r(n-1)$ est bien le $pgcd(a, b)$:
   - En effet, soit d un diviseur commun à a et à b alors il divise $r(1)$ car $r(1) = a - b\ q(0)$; de même puisqu'il divise b et $r(1)$ il divisera $r(2)$ car $r(2) = b - r(1)\ q(1)$ et ainsi de suite, il finira par diviser $r(n-1)$. Par conséquent tout diviseur de a et de b divise donc $r(n-1)$.
   - Réciproquement, on démontre de même, qu'un diviseur de $r(n-1)$ divise $r(n-2)$ (à cause de la dernière division) ; s'il divise $r(n-1)$ et $r(n-2)$ ill divisera $r(n-3)$ à cause de l'avant-dernière division , etc ..., il divisera donc b et donc a. Par conséquent, le plus grand diviseur de $r(n-1)$ est le plus grand commun diviseur de a et de b. Or le plus grand diviseur de $r(n-1)$ est $r(n-1)$ lui-même.
   - Ainsi $pgcd(a, b) = r(n-1)$

4. Soit Fib la suite de Fibonacci définie par $Fib(0) = 1$; $Fib(1) = 1$ et $Fib[n] = Fib[n-1] + Fib[n-2]$ pour tout entier $n \geq 2$

   (a) On pose Φ = le nombre d'Or = $\dfrac{1+\sqrt{5}}{2}$ alors $\forall n \in \mathbb{N}\ \Phi^{n-1} \leq Fib(n)$.
   Démontrons-le :
   Comme Fib est une suite doublement récurrente, on considère son équation caractéristique $q^2 = q + 1$ qui a pour solutions Φ et et $\Psi = \dfrac{-1}{\Phi} = \dfrac{1-\sqrt{5}}{2}$.
   alors pour tout entier naturel n l'on a $Fib(n) = a\Phi^n + b\Psi^n$. Comme $Fib(0) = 1$ et $Fib(1) = 1$ alors a et b vérifient le système

$$\begin{cases} 1 = a\Phi^0 + b\Psi^0 \\ 1 = a\Phi^1 + b\Psi^1 \end{cases}$$

   donc

$$\begin{cases} 1 = a + b \\ 1 = a\Phi + b\Psi \end{cases}$$

   d'où

$$\begin{cases} b = 1 - a+ \\ 1 = a\Phi + (1-a)\Psi \end{cases}$$

   On a donc

$$\begin{cases} b = 1 - a \\ a = \dfrac{1-\Psi}{\Phi + \Psi} \end{cases}$$

   d'où

$$\begin{cases} b = 1 - a \\ a = \dfrac{1-\Psi}{\sqrt{5}} \end{cases}$$

   On en tire que

$$\begin{cases} b = \dfrac{\Phi - 1}{\sqrt{5}} \\ a = \dfrac{1-\Psi}{\sqrt{5}} \end{cases}$$

   Par récurrence, l'on prouve alors que $\forall n \in \mathbb{N}\ \Phi^{n-1} \leq Fib(n)$**.**

   - Etape 1 : Cette propriété est vraie pour $n = 0$ car $\Phi^{0-1} = \dfrac{1}{\phi} = \dfrac{2}{1+\sqrt{5}} \leq Fib(0) = 1$
     en effet $\sqrt{5} \approx 1,623$ donc $1 + \sqrt{5} \approx 2,623$ donc $\dfrac{2}{1+\sqrt{5}} \leq 1$
   - Etape 2 : soit n un entier naturel fixé, supposons que pour tout $k \leq n$ l'on a $\Phi^{n-1} \leq Fib(n)$, démontrons qu'alors $\Phi^n \leq Fib(n+1)$
     Comme Φ vérifie l'équation caractéristique $\Phi^2 = \Phi + 1$ donc $\Phi^2\Phi^{n-2} = \Phi\Phi^{n-2} + 1\Phi^{n-2}$ donc $\Phi^n = \Phi^{n-1} + \Phi^{n-2} \leq Fib(n) + Fib(n-1)$ donc $\Phi^n \leq Fib(n+1)$.
   - Etape 3 : d'après étape 1 et étape 2, on peut conclure que $\forall n \in \mathbb{N}\Phi^{n-1} \leq Fib(n)$

   (b) **Démontrons par récurrence sur** k **que** $r(n-k-1) \geq Fib(k)$
   - Etape 1 : Cette propriété est vraie pour $k = 0$ car $r(n-1) \geq Fib(0) = 1$
     en effet $r(n-1)$ est le dernier reste non nul donc $r(n-1) \geq 1$

- Etape 2 : supposons que pour $r(n-1) \geq Fib(0)$ et $r(n-2) \geq Fib(1)$ et $\cdots$ et $r(n-k-1) \geq Fib(k)$
  Démontrons qu'alors $r(n-k-2) \geq Fib(k+1)$
  Or $r(n-k-2) = q(n-k-1)r(n-k-1)+r(n-k)$ avec $0 \leq r(n-k) < r(n-k-1)$
  donc $r(n-k-2) = q(n-k-1)r(n-k-1)+r(n-k) \geq q(n-k-1)Fib(k)+Fib(k-1)$
  Comme les quotients ne peuvent s'annuler car alors 2 restes seraient identiques ce qui ne se peut car la suite des restes est strictement décroissante alors $q(n-k-1) \geq 1$
  Donc $r(n-k-2) \geq Fib(k)+Fib(k-1)$ donc $r(n-k-2) \geq Fib(k+1)$
- Etape 3 : d'après étape 1 et étape 2, on peut conclure que $\forall k \in \mathbb{N}\ r(n-k-1) \geq Fib(k)$

(c) On en déduit que $n \leq Aln(a)+B$.
Comme $\forall k \in \mathbb{N}\ r(n-k-1) \geq Fib(k)$ donc pour $k=n$ l'on a $r(-1) \geq Fib(n)$ donc $a \geq Fib(n)$ Or $Fib(n) \geq \Phi^{n-1}$ donc $a \geq \Phi^{n-1}$ donc $ln(a) \geq (n-1)ln(\Phi)$ donc $n-1 \leq \dfrac{1}{ln(\phi)}ln(a)$
Donc $n \leq 1+\dfrac{1}{ln(\phi)}ln(a)$
Par exemple pour $a=44$ on a théoriquement $n \leq 1+\dfrac{1}{ln(\phi)}ln(44) \approx 8,86$ alors qu'en pratique $n=3$

$$L(a;b) \leq Aln(a)+B$$

donc la complexité en temps de cet algorithme au pire est en $O(ln(a))$

6.4 Exercice : Calcul de la valeur d'un polynôme en une valeur

On dit que deux fonctions f et g de $\mathbb{N}^*$ dans $\mathbb{R}^+$ ont même ordre de grandeur asymptotique (ce que l'on note $f(n) = \Theta(g(n))$ lorsqu'il existe un réel $C > 0$ et un réel $D > 0$ ainsi qu'un entier n_0 tel que pour tout entier $n \geq n_0$ l'on ait

$$Cg(n) \leq f(n) \leq Dg(n)$$

6.4.1 Méthode classique

Soit le polynôme $P(x) = 3x^4 - 5x^3 + 6x^2 - 2x + 4$
Pour calculer $P(a)$ il faut normalement 4 additions et 10 multiplications.
Pour le polynôme de degré n : $P(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1} + a_nx^n$ le coût maximum du calcul de $P(a)$ en opérations élémentaires(additions et multiplications) est de :
n additions et de $1 + 2 + \cdots . + n$ c'est-à-dire $\dfrac{n(n+1)}{2}$ multiplications.
Le coût total maximum est $MAX_1(n) = n + \dfrac{n(n+1)}{2} = \dfrac{n^2+3n}{2}$.
$\lim_{n\mapsto+\infty} MAX_1(n) = \lim_{n\mapsto+\infty} n + \dfrac{n(n+1)}{2} = \lim\limits_{n\mapsto+\infty} \dfrac{n^2}{2}$
De plus dès que $n \geq 1$ on a $n \leq n^2$ donc $0 \leq \dfrac{3n}{2} \leq \dfrac{3n^2}{2}$
donc $0 + \dfrac{n^2}{2} \leq \dfrac{3n+n^2}{2} \leq \dfrac{3n^2}{2} + \dfrac{n^2}{2}$
d'où $\dfrac{1}{2}n^2 \leq MAX_1(n) \leq 2n^2$ donc $MAX_1(n) = \Theta(n^2)$.
Cet algorithme classique de calcul de $P(a)$ a pour ordre de grandeur n^2.

6.4.2 Méthode de Horner

HORNER (mathématicien et physicien anglais 1786-1837) a mis en forme une méthode proposée 150 ans plus tôt par Newton Il propose l'algorithme suivant :
$3x^4 - 5x^3 + 6x^2 - 2x + 4 = (((3x - 5)x + 6)x - 2)x + 4$
On se retrouve avec toujours 4 additions mais seulement 4 multiplications.
De façon générale, $a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1} + a_nx^n = (((((a_nx + a_{n-1})x + a_{n-2})x + \ldots\ldots\ldots + a_1)x + a_0$
L'algorithme de Horner coûte au maximum n additions et n multiplications donc le coût total maximum est $MAX_2(n) = n + n = 2n$
$\lim_{n\mapsto+\infty} MAX_2(n) = \lim_{n\mapsto+\infty} 2n$
Pour tout entier naturel n, l'on a $n \leq 2n \leq 3n$ donc $MAX_2(n) = \Theta(n)$
donc l'algorithme de Horner de calcul de $P(a)$ a pour ordre de grandeur n.
Il est plus performant donc que l'algorithme classique.
Reprenons l'exemple $P(x) = 3x^4 - 5x^3 + 6x^2 - 2x + 4$ ici $a_4 = 3$; $a_3 = -5$; $a_2 = 6$; $a_1 = -2$; $a_0 = 4$ d'après Horner $P(x_0) = (((3x_0 - 5)x_0 + 6)x_0 - 2)x_0 + 4$ On pose :

- $h_4 = a_4$
- $h_3 = h_4x_0 + a_3$
- $h_2 = h_3x_0 + a_2$
- $h_1 = h_2x_0 + a_1$

- $h_0 = h_1 x_0 + a_0$

donc $P(x_0) = h_0$
On peut disposer les calculs selon le schéma suivant appelé shéma de Horner

a_4	a_3	a_2	a_1	a_0
	$+x_0 h_4$	$+x_0 h_3$	$+x_0 h_2$	$+x_0 h_1$
$= h_4$	$= h_3$	$= h_2$	$= h_1$	$= h_0 = P(x_0)$

Pour notre exemple, prenons $x_0 = 2$

3	-5	6	-2	4
	$+2 \times 3$	$+2 \times 1$	$+2 \times 8$	$+2 \times 14$
$= 3$	$= 1$	$= 8$	$= 14$	$= 32 = P(2)$

1. **Théorème de Horner**
   $P(x) = (x - x_0)(h_4 x^3 + h_3 x^2 + h_2 x + h_1) + h_0$
2. **Corollaire** Si x_0 est une racine de $P(x)$ c'est-à-dire que $P(x_0) = 0 = h_0$ alors $P(x) = (x - x_0)(h_4 x^3 + h_3 x^2 + h_2 x + h_1)$

6.5 Complexité du pivot de Gauss - Centrale MP 2008

Soit un entier $n \geq 2$. Soit $\mathcal{M}_n$ l'ensemble des matrices carrées à n lignes, à coefficients réels. On note $\mathcal{TS}_n \subset \mathcal{M}_n$ l'ensemble des matrices triangulaires supérieures
Soit la matrice $A = (a_{ij}) \in \mathcal{TS}_n$.
On considère le système linéaire (1) : $Au = w$ où $w =^t (w_1, w_2, \cdots, w_n)$ est donné et $u =^t (u_1, u_2, \cdots, u_n)$ est l'inconnue.
On suppose que $det(A) \neq 0$ donc (1) admet une unique solution $u =^t (u_1, u_2, \cdots, u_n) \in \mathbb{R}^n$.

1. Calculer u_n puis pour tout $k \in [|1; n-1|]$ exprimer u_{n-k} en fonction de $u_n, u_{n-1}, \cdots, u_{n-k+1}$
   Travaillons d'abord sur un exemple $n = 3$.

$$\begin{cases} a[1,1]u_1 + a[1,2]u_2 + a[1,3]u_3 = w_1 \\ a[2,2]u_2 + a[2,3]u_3 = w_2 \\ a[3,3]u_3 = w_3 \end{cases}$$

$$\iff \begin{cases} a[1,1]u_1 + a[1,2]u_2 + a[1,3]u_3 = w_1 \\ a[2,2]u_2 + a[2,3]u_3 = w_2 \\ u_3 = \dfrac{1}{a[3,3]}w_3 \end{cases}$$

$$\iff \begin{cases} a[1,1]u_1 + a[1,2]u_2 + a[1,3]u_3 = w_1 \\ a[2,2]u_2 = w_2 - a[2,3]u_3 \\ u_3 = \dfrac{1}{a[3,3]}w_3 \end{cases}$$

$$\iff \begin{cases} a[1,1]u_1 + a[1,2]u_2 + a[1,3]u_3 = w_1 \\ u_2 = \dfrac{1}{a[2,2]}(w_2 - a[2,3]u_3) \\ u_3 = \dfrac{1}{a[3,3]}w_3 \end{cases}$$

$$\iff \begin{cases} a[1,1]u_1 = w_1 - a[1,2]u_2 - a[1,3]u_3 \\ u_2 = \dfrac{1}{a[2,2]}(w_2 - a[2,3]u_3) \\ u_3 = \dfrac{1}{a[3,3]}w_3 \end{cases}$$

$$\iff \begin{cases} u_1 = \dfrac{1}{a[1,1]}(w_1 - a[1,2]u_2 - a[1,3]u_3) \\ u_2 = \dfrac{1}{a[2,2]}(w_2 - a[2,3]u_3) \\ u_3 = \dfrac{1}{a[3,3]}w_3 \end{cases}$$

- Pour $u[3]$ on a 1 division, 0 soustractions et 0 multiplications
- Pour $u[2]$ on a 1 division, 1 soustractions et 1 multiplications

- Pour $u[1]$ on a 1 division, 2 soustractions et 2 multiplications
- donc en tout on a 3 divisions, $0+1+2$ soustractions et $0+1+2$ multiplications

On peut donc généraliser pour n quelconque.
Les solutions du système sont :

$$\begin{cases} u_n = \dfrac{1}{a[n,n]} w_n \\ \forall k \in [|1; n-1|]\ u_{n-k} = \dfrac{1}{a[n-k,n-k]}(w_{n-k} - \displaystyle\sum_{i=n-k+1}^{n} a[n-k,i]u_i) \end{cases}$$

2. On peut écrire ceci sous forme d'un algorithme de résolution codé en Maple

```
   n:=3;
w[1] := 44;w[2] := 26;w[3] := 30;
a[1,1] :=1;a[1,2] := 3; a[1,3] := 5;
a[2,2] := 2 ; a[2,3] := 3.
a[3,3] := 5;
    u[n]= (1/a[n,n])*w[n];
 for k from 1 to n - 1
 do
  u[n - k] =(1/a[n-k,n - k])[w[n-k] - sum(a[n - k, i] * u[i], i = n - k + 1..n)];
 print(u[n-k]);
od;
```

Ce programme affiche $u[1] = 2; u[2] = 4; u[3] = 6$
On a bien

$$\begin{cases} 1 \times 2 + 3 \times 4 + 5 \times 6 = 44 \\ 2 \times 4 + 3 \times 6 = 26 \\ 5 \times 6 = 30 \end{cases}$$

3. - Pour $u[n]$ on a 1 division, 0 soustractions et 0 multiplications
   - Pour $u[n-1]$ on a 1 division, 1 soustractions et 1 multiplications
   - Pour $u[n-2]$ on a 1 division, 2 soustractions et 2 multiplications
   - $\cdots$
   - Pour $u[1]$ on a 1 division,$n-1$ soustractions et $n-1$ multiplications
   - donc en tout on a :
     n divisions, $0+1+2+\cdots+(n-1) = \dfrac{(n-1)n}{2}$ soustractions et $0+1+2+\cdots+(n-1) = \dfrac{(n-1)n}{2}$ multiplications

Le coût total de l'algorithme du pivot de Gauss est donc :
$n + 2\dfrac{(n-1)n}{2} = n + n^2 - n = n^2$.
Cet algorithme est donc un algorithme dont l'ordre de grandeur asymptotique est n^2.

6.6 Edhec 1996

Partie 1

On considère f la fonction numérique d'une variable réelle définie par :
$f(x) = x - \ln(x)$

1. Etudier f et résumer cette étude par un tableau de variations.
2. Etudier le signe de $f(x) - x$ pour $x > 0$

Partie 2

On considère l'algorithme suivant :

```
program iter;
var   n , k : integer ;
      a , u , p : real ;
 function f(x : real): real;
      begin
            if x > 0 then f := x - ln(x) ;
      end;
 begin
   (* entrée des données *)
        readln(n);
        readln(a);

   (* traitement *)
       u := a ;
       p := a;
       for k := 1 to n do     begin
                                   u := f(u) ;
                                   p:= p* u;
                               end;

   (* affichage des résultats *)
       writeln(u);
       writeln(p);
 end.
```

Dans le cas particulier où $n = 3$ et $a = 2$

1. Donner les écritures des valeurs exactes des contenus de la variable u à chaque tour de boucle et à la fin de l'algorithme.
2. On donne $2 - \ln(2) \approx 1,306$
   $2 - \ln(2) - \ln(2 - \ln(2)) \approx 1,039$
   $2,612 * 1,039 \approx 2,714$
   $2 - \ln(2) - \ln(2 - \ln(2)) - \ln(2 - \ln(2) - \ln(2 - \ln(2))) \approx 1,00074$
   $2,715 * 1,00074 \approx 2,716$
   Donner les valeurs approchées des contenus successifs de la variable p à chaque tour de boucle et à la fin de l'algorithme. Que peut-on conjecturer pour cette variable p ?

Partie 3

Dorénavant, dans le cas général, on note u_n et p_n les contenus respectifs des variables u et p à la fin de l'algorithme lorsque leur calcul est possible.

1. Pour quelles valeurs de a peut-on définir cette suite (u_n) de premier terme $u_0 = a$ et dont le terme général est calculé par l'algorithme précédent ?
2. Pour les valeurs de a trouvées ci-dessus, donner en fonction de n :
   - le nombre total d'appels de la fonction f utilisée dans l'algorithme
   - le nombre total de soustractions nécessaires aux calculs de u et de p
   - le nombre total de multiplications nécessaires aux calculs de u et de p
   - le nombre total d'affectations nécessaires aux calculs de u et de p. a

   NB - On admettra que le symbole := utilisé dans l'écriture $for\ k := 1\ to\ n$ ne sera pas considéré comme une affectation mais que chaque appel de fonction nécessite une affectation ($f :=$) et une soustraction.
3. Lorsque la suite (u_n) est bien définie, écrire , pour tout entier naturel n, la relation liant u_{n+1} et u_n.
4. Démontrer que si $a = 1$ alors la suite (u_n) est constante.
5. Démontrer que si la suite (u_n) est constante alors $a = 1$
6. On suppose dans cette question que $a > 1$.
   (a) Montrer que $\forall n \in \mathbb{N}$, l'on a : $u_n > 1$
   (b) Etudier les variations de la suite (u_n)
   (c) En déduire que la suite (u_n) converge puis déterminer sa limite.
7. On suppose dans cette question que $0 < a < 1$.
   (a) Montrer que $\forall n \in \mathbb{N}^*$, l'on a : $u_n > 1$
   (b) Etudier les variations de la suite (u_n) sur $\mathbb{N}^*$
   (c) En déduire que la suite (u_n) converge puis déterminer sa limite.

Partie 4

1. Démontrer par récurrence que $\forall n \in \mathbb{N}^*\ p_n = au_1u_2 \cdots u_n$
2. En considérant $\ln(p_n)$, montrer que $\forall n \in \mathbb{N}^*\ p_n = e^{a-u_{n+1}}$
3. En déduire $\lim\limits_{n\to+\infty} p_n$
4. Peut-on alors justifier la conjecture de la question 2 de la partie 2 lorsque a valait 2 ?
5. Ecrire alors un nouvel algorithme en Turbo-Pascal permettant le calcul de p_n.
   Cet algorithme ne doit contenir aucune multiplication.
   On rappelle que la fonction exponentielle notée exp est prédéfini dans le langage Turbo-Pascal.

6.6.1 Corrigé

Dans le cas particulier où $n = 3$ et $a = 2$

1. **Donner les écritures des valeurs exactes des contenus de la variable u à chaque tour de boucle et à la fin de l'algorithme.**

contenus successifs des cases	k	n	a	u	p
entree des donnees		3			
			2		
avant entree boucle				2	
					2
tour de boucle	1			2 - ln(2)	
					2 * (2 - ln(2))
tour de boucle	2			2 - ln(2) - ln(2 - ln(2))	
					2*(2 - ln(2) * (2 - ln(2) - ln(2 - ln(2)))
tour de boucle	3			2 - ln(2) - ln(2 - ln(2))) - ln(2 - ln(2) - ln(2 - ln(2))))	
					[2*(2 - ln(2) * (2 - ln(2) - ln(2 - ln(2)))] * [2 - ln(2) - ln(2 - ln(2))) - ln(2 - ln(2) - ln(2 - ln(2))))]
sortie de boucle				affichage du contenu de cette case	
					affichage du contenu de cette case

2. **On donne** $2 - \ln(2) \approx 1,306$ **;** $2 - \ln(2) - \ln(2 - \ln(2)) \approx 1,039$ **;** $2,612 * 1,039 \approx 2,714$ **;** $2 - \ln(2) - \ln(2 - \ln(2)) - \ln(2 - \ln(2) - \ln(2 - \ln(2))) \approx 1,00074$ **;** $2,715 * 1,00074 \approx 2,716$ **;** **Donner les valeurs approchées des contenus successifs de la variable p à chaque tour de boucle et à la fin de l'algorithme. Que peut-on conjecturer pour cette**

variable p ?

valeur approchee de	u	p
	2	2
$k=1$	$1,306852819$	$2,613705639$
$k=2$	$1,039231001$	$2,716243928$
$k=3$	$1,000749983$	$2,718281065$

On peut donc conjecturer que la suite (p_n) semble converger vers $e \approx 2,718$

Partie 3

Dorénavant, dans le cas général, on note u_n et p_n les contenus respectifs des variables u et p à la fin de l'algorithme lorsque leur calcul est possible.

1. **Pour quelles valeurs de a peut-on définir cette suite (u_n) de premier terme $u_0 = a$ et dont le terme général est calculé par l'algorithme précédent ?** A condition de prendre $a > 0$ Le programme simule l'affichage du nième terme des deux suites :
   - (u_n) définie par $u_0 = a$ et $\forall n \in \mathbb{N}\ u_{n+1} = f(u_n)$
   - (p_n) définie par $p_0 = a$ et $\forall n \in \mathbb{N}\ p_{n+1} = u_n p_n$
2. **NB - On admettra que le symbole $:=$ utilisé dans l'écriture $for\ k := 1\ to\ n$ ne sera pas considéré comme une affectation mais que chaque appel de fonction nécessite une affectation $(f :=)$ et une soustraction.**
   Pour les valeurs de a trouvées ci-dessus, donner en fonction de n :
   - **le nombre total d'appels de la fonction f utilisée dans l'algorithme**
     C'est n car à chacun des n tours de boucle $for\ k := 1\ to\ n$, on appelle une seule fois la fonction f par $u := f(u)$.
   - **le nombre total de soustractions nécessaires aux calculs de u et de p**
     C'est n car à chacun des n tours de boucle, on réalise une seule soustraction située à l'intérieur de la fonction f.
   - **le nombre total de multiplications nécessaires aux calculs de u et de p**
     C'est n car à chacun des n tours de boucle, on réalise une seule multiplication $p := p*u$.
   - **le nombre total d'affectations nécessaires aux calculs de u et de p.**
     C'est $2 + 3n$ car il y a deux affectations avant d'entrer dans la boucle $u := a$ et $p := a$ puis à chacun des n tours de boucle, on réalise 3 affectations la première est $u := f(u)$; la seconde est située dans le calcul de $f(u)$ et la troisième est $p := p * u$.
3. **Lorsque la suite (u_n) est bien définie, écrire , pour tout entier naturel n, la relation liant u_{n+1} et u_n.**
   (u_n) définie par $u_0 = a$ et $\forall n \in \mathbb{N}\ u_{n+1} = f(u_n) = u_n - \ln(u_n$
4. **Démontrer que si $a = 1$ alors la suite (u_n) est constante.**
   Démontrons par récurrence que $\forall n \in \mathbb{N}\ u_n = 1$:
   - cette propriété est vraie pour $n = 0$ car $u_0 = a = 1$
   - soit k un entier naturel. supposons que $u_k = 1$ alors $u_{k+1} = f(u_k) = u_k - ln(u_k) = 1 - \ln(1) = 1$
   - La propriété étant initialisée en 0 et héréditaire alors elle est vraie pour tout entier naturel n.
5. **Démontrer que si la suite (u_n) est constante alors $a = 1$**
   (u_n) constante $\Leftrightarrow \forall n \in \mathbb{N}\ u_{n+1} = u_n \Leftrightarrow \forall n \in \mathbb{N}\ u_n - \ln(u_n) = u_n \Leftrightarrow \forall n \in \mathbb{N}\ \ln(u_n) = 0$
   $\Leftrightarrow \forall n \in \mathbb{N}\ u_n = 1$
   Donc si la suite (u_n) est constante alors $u_0 = 1$ donc $a = 1$

6. **On suppose dans cette question que** $a > 1$**.**
   (a) **Montrer que** $\forall n \in \mathbb{N}$ **, l'on a :** $u_n > 1$
   Démontrons par récurrence que $\forall n \in \mathbb{N}\ u_n > 1$:
   - cette propriété est vraie pour $n = 0$ car $u_0 = a > 1$
   - soit k un entier naturel. supposons que $u_k > 1$ alors $f(u_k) > f(1)$ car f est croissante sur $]1; +\infty[$. Or $u_{k+1} = f(u_k)$ et $f(1) = 1$ donc $u_{k+1} > 1$
   - La propriété étant initialisée en 0 et héréditaire alors elle est vraie pour tout entier naturel n.

   (b) **Etudier les variations de la suite** (u_n)
   $\forall n \in \mathbb{N}\ u_{n+1} - u_n = -\ln(u_n) < 0$ car $\forall n \in \mathbb{N}$, l'on a : $u_n > 1$ donc $\ln(u_n) > 0$. Par conséquent la suite (u_n) est décroissante strictement sur $\mathbb{N}$

   (c) **En déduire que la suite** (u_n) **converge puis déterminer sa limite.**
   - La suite (u_n) est décroissante et minorée par 1 donc elle converge vers L.
   - Comme $\forall n \in \mathbb{N}\ u_n > 1$ donc $L > 1$
   - Comme $L > 1$ et que f est continue sur $]1; +\infty[$ alors f est continue en L
   - Comme $u_{n+1} = f(u_n)$
   - A cause de l'unicité de la limite, L vérifie donc l'équation suivante : $L = f(L)$ donc $L - \ln(L) = L$ donc $\ln(L) = 0$ donc $L = 1$

7. **On suppose dans cette question que** $0 < a < 1$**.**
   (a) **Montrer que** $\forall n \in \mathbb{N}^*$ **, l'on a :** $u_n > 1$ Démontrons par récurrence que $\forall n \in \mathbb{N}^*$ $u_n > 1$:
   - cette propriété est vraie pour $n = 1$ car $u_1 = f(u_0) = f(a) > 1$ car $0 < a < 1$ et que f est décroissante sur $]0; 1[$ et que $f(1) = 1$
   - soit k un entier naturel non nul. Supposons que $u_k > 1$ alors $f(u_k) > f(1)$ car f est croissante sur $]1; +\infty[$. Or $u_{k+1} = f(u_k)$ et $f(1) = 1$ donc $u_{k+1} > 1$
   - La propriété étant initialisée en 1 et héréditaire alors elle est vraie pour tout entier naturel n non nul.

   (b) **Etudier les variations de la suite** (u_n) **sur** $\mathbb{N}^*$ $\forall n \in \mathbb{N}^*\ u_{n+1} - u_n = -\ln(u_n) < 0$ car $\forall n \in \mathbb{N}^*$, l'on a : $u_n > 1$ donc $\ln(u_n) > 0$. Par conséquent la suite (u_n) est décroissante strictement sur $\mathbb{N}^*$

   (c) **En déduire que la suite** (u_n) **converge puis déterminer sa limite.**
   - La suite (u_n) est décroissante sur $\mathbb{N}^*$ et minorée par 1 donc elle converge vers L.
   - Comme $\forall n \in \mathbb{N}^*\ u_n > 1$ donc $L > 1$
   - Comme $L > 1$ et que f est continue sur $]1; +\infty[$ alors f est continue en L
   - Comme $u_{n+1} = f(u_n)$
   - A cause de l'unicité de la limite, L vérifie donc l'équation suivante : $L = f(L)$ donc $L - \ln(L) = L$ donc $\ln(L) = 0$ donc $L = 1$

Partie 4

1. Démontrer par récurrence que $\forall n \in \mathbb{N}^*\ p_n = au_1u_2 \cdots u_n$
   - Avant d'entrer dans la boucle u contient a et p contient a, donc $u_0 = a$ et $p_0 = a$. Puis au premier tour de boucle u reçoit $f(u_0) = u_1$ puis la case p reçoit $p * u$ donc si $n = 1$ on a $p_1 = p_0u_1 = au_1$
   - soit k un entier naturel. Supposons que $p_k = au_1u_2 \cdots u_k$ alors au tour de boucle suivant $k + 1$ la case u contient u_{k+1} alors l'instruction $p := p * u$ crée alors $p_{k+1} = (au_1u_2 \cdots u_k)(u_{k+1})$
   - La propriété étant initialisée en 1 et héréditaire alors elle est vraie pour tout entier naturel n non nul.

2. **En considérant** $\ln(p_n)$**, montrer que** $\forall n \in \mathbb{N}^*\ p_n = e^{a-u_{n+1}}$
   Comme $\forall n \in \mathbb{N}^*\ p_n = au_1u_2\cdots u_n$ alors $\forall n \in \mathbb{N}^*\ \ln(p_n) = \ln(au_1u_2\cdots u_n) = \ln(a) + \ln(u_1) + \cdots + \ln(u_n)$
   Donc

$$\left|\begin{array}{cc} u_{n+1} - u_n = & -\ln(u_n) \\ u_n - u_{n-1} = & -\ln(u_{n-1}) \\ u_{n-1} - u_{n-2} = & -\ln(u_{n-2}) \\ & \\ u_3 - u_2 = & -\ln(u_2) \\ u_2 - u_1 = & -\ln(u_1) \\ u_1 - u_0 = & -\ln(u_0) \end{array}\right|$$

   Alors par sommation télescopique on obtient :
   $u_{n+1} - u_0 = -\ln(p_n)$ donc $u_{n+1} - a = -\ln(p_n)$ donc $\ln(p_n) = a - u_{n+1}$ On en déduit que $\forall n \in \mathbb{N}^*\ p_n = e^{a-u_{n+1}} = \dfrac{e^a}{e^{u_{n+1}}}$
3. **En déduire** $\lim\limits_{n\to+\infty} p_n$
   Quand $n \to +\infty$ u_{n+1} tend vers 1 donc $a - u_{n+1}$ tend vers $a - 1$ donc $e^{a-u_{n+1}}$ tend vers $e^{a-1} = \dfrac{e^a}{e}$
4. **Peut-on alors justifier la conjecture de la question** 2 **de la partie** 2 **lorsque** a **valait** 2 **?**
   Lorsque $a = 2$ alors $a - 1 = 1$ donc $\lim\limits_{n\to+\infty} p_n = e \approx 2,718$
5. **Ecrire alors un nouvel algorithme en Turbo-Pascal permettant le calcul de** p_n**. Cet algorithme ne doit contenir aucune multiplication.**

```
program iter;
var   n , k : integer ;
      a , u , p : real ;
 function f(x : real): real;
      begin
            if x > 0 then f := x - ln(x) ;
      end;
 begin
   (* entrée des données *)
        readln(n);
        readln(a);

   (* traitement *)
       u := a ;
       for k := 1 to n + 1 do  begin
                                              u := f(u) ;
                                  end;
       p :=exp(a)/exp(u);
   (* affichage des résultats *)
       writeln(u);
       writeln(p);
 end.
```

Chapitre 7

Programmation et langages

7.1 Langages

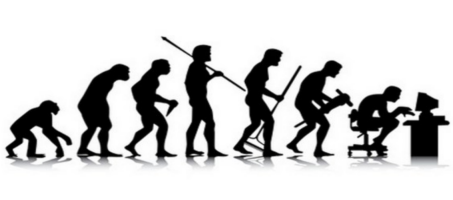
7.1.1

7.1.2

7.2 Langages non algorithmiques

Chapitre 8

Informatique numérique



8.1 Historique

Les ordinateurs sont nés, avant la deuxième guerre mondiale, de la nécessité d'effectuer des calculs d'où leur premier nom : les calculateurs. Ce n'est qu'en 1956 que le français Jacques PERRET définit pour la société américaine IBM le nom d'ordinateur.
La création des calculateurs a permis d'élaborer une nouvelle science avec ses méthodes et ses techniques particulières qui se détachera des Mathématiques pour devenir L'Informatique ;
L'informatique numérique est une branche de l'informatique concernant les calculs numériques effectués à l'aide d'une calculatrice ou d'un ordinateur.
Un algorithme (du nom du mathématicien arabe AL KURAWISMI) est un ensemble de règles qui donne un résultat à partir d'une situation donnée.Chaque étape est soigneusement définie pour que sa traduction soit claire en langage informatique et réalisable par l'ordinateur.
Parmi les algorithmes, on peut distinguer une catégorie particulière : les algorithmes numériques dont les données et les résultats sont des nombres.
Il y a une idée de base qu'il faut toujours garder à l'esprit lorsque l'on effectue du calcul sientifique à l'aide d'une machine :

1. Etre toujours critique face à un résultat brut fourni par un ordinateur.
2. Le résultat affiché n'est pas forcément celui qui est représenté en mémoire.
3. Les calculs peuvent être entachés d'erreurs de deux types :
   (a) Les erreurs de méthode dues à l'algorithme utilisé, indépendantes de lamachine utilisée.
   (b) Les erreurs dues à la représentation en mémoire et à leur manipulation.

8.2 Ecriture en virgule flottante

8.2.1 Définition

C'est par des entiers que l'on représente le mieux beaucoup de nombres.
Les entiers sont simples à comprendre, à manipuler, à utiliser en représentation binaire.
Toutefois, ils ne conviennent pas pour
1 - les très grandes valeurs : un entier très grand se termine par une suite impressionnante de zéros.
Par exemple, la distance Terre-Soleil : 150000000 km
2 - les nombres rationnels
En Mathématiques, dans ces 2 cas on représente ces nombres à l'aide d'une virgule.
$150\ 000\ 000 = 1,5 \times 10^8 = 15, \times 10^7 = 150, \times 10^6 = \cdots$
$1 + \frac{1}{2} = 1,5$
Multiplier par 10 revient à déplacer la virgule d'un rang vers la droite.
Diviser par 10 revient à déplacer la virgule d'un rang vers la gauche.
La virgule "flotte". Cette écriture est appelée écriture en virgule flottante.
En virgule flottante, 2 parties représentent un nombre :
1 - la mantisse : $1 \leq mantisse < 10$
2 - l'exposant qui est un entier relatif.
Par exemple, $9,8 \times 10^7$ a pour mantisse $9,8$ et pour exposant 7
$1,3 \times 10^{-1}$ a pour mantisse $1,3$ et pour exposant -1

8.2.2 Représentation binaire en virgule flottante

Elle est semblable à la représentation en virgule flottante en base 10.
Un nombre s'écrit en 2 parties : la mantisse $m \in [1;2[$ et l'exposant e qui est une puissance de 2. On a besoin du tableau suivant :

Valeur binaire	Valeur décimale
1	2
0	0
$0,1$	$\frac{1}{2}$
$0,01$	$\frac{1}{4}$
$0,001$	$\frac{1}{8}$
$0,0001$	$\frac{1}{16}$
$0,00001$	$\frac{1}{32}$
$0,000001$	$\frac{1}{64}$

Exemple

$\overline{1,101} \times \overline{10}^{\overline{100}} = \left(1 + 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8}\right) \times 2^4 = \left(1 + \frac{5}{8}\right) \times 16 = 1,625 \times 16 = 26$

Autre méthode :

$\overline{1,101} \times \overline{10}^{\overline{100}} = \overline{1,101} \times 2^4$ donc on déplace la virgule de 4 rangs vers la droite : on obtient $\overline{11010} = 26$

8.3 Stockage en mémoire

Le nombre à virgule flottante est stockée dans une zone qui comprend 2 champs :

- Champ 1 : signe et valeur de la mantisse.
- Champ 2 : signe et valeur de l'exposant.

La dimension du champ mantisse détermine la précision du nombre .
Exemple :

- En base 10, $1,234$ est plus précis que $1,2$
- En base 2, $\overline{1,000}$ est plus précis que $\overline{1,0}$
- Les zéros en tête d'un nombre ne contribuent pas à sa précision.

En Turbo-Pascal de BORLAND, un réel x est codé sur 6 octets :

Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
exposant					

- L'exposant e est donc codé sur 8 bits donc offre $2^8 = 256$ configurations.
  Par conséquent, $0 \leq e \leq 255$. Cas spécial : si $e = 0$ alors $x = 0$
- Pour les 5 octets suivant offrant $5 \times 8 = 40$ bits :
  - Le bit le plus à gauche de l'octet 1 est le bit de signe : 0 si $x \geq 0$ sinon 1
  - Il reste alors 39 bits pour coder la mantisse m donc $0 \leq m < 2^{39}-1 = 549\,755\,813\,887$
  - Soit un réel x alors $x = m \times E^e$ est codé $m \times 10^e$
  - Comme on veut que le plus petit réel informatique soit l'inverse du plus grand réel informatique alors la formule de
    $|x| = (2^{e-168})(2^{39} + m)$ si $x \neq 0$.
    Bien entendu, 0 a pour exposant $e = 0$ et pour mantisse $m = 0$
  - Le plus petit réel informatique a pour exposant $e = 1$ et une mantisse$m = 0$ donc
    $min = (2^{1-168})(2^{39} + 0) = 2^{-167}2^{39} = 2^{-128} = \dfrac{1}{2^{128}} \approx 0,2938735877055721 \times 10^{-38}$
  - Le plus petit grand réel informatique a pour exposant $e = 255$ et une mantisse$m = 2^{39} - 1$ donc
    $min = (2^{255-168})(2^{39}+2^{39}-1) = 2^{87}(2^{40}-1) \approx 2^{87}\ 2^{40} \approx 2^{127} \approx 1,7014183460469 \times 10^{38}$

En BASIC MICROSOFT simple précision

Octet 0	Octet 1	Octet 2	Octet 3
exposant			

$2^{128} + 24 = 152$ bits donc $|x| = 2^{e-152}(2^{23} + 1)$
En BASIC MICROSOFT double précision

Octet 0	Octet 1	Octet 2	Octet 3	$textOctet4$	$textOctet5$	Octet 6	$textOctet7$
exposant							

$2^{128} + 56 = 174$ bits donc $|x| = 2^{e-174}(2^{55} + 1)$

8.3.1 Exemple : les lapins de Fibonacci

```
var A,B,I,N : integer;
       NBCOUPLES : real;
begin
(* entrée des données *)
repeat
write(' Tapez un nombre de mois N = ');
readln(N);
until ((N = int(N)) and (N > 0))

(* traitement *)
A := 1; B := 1;
if N <=2 then NBCOUPLES := 1
                else begin
                 I := 3 ;
while I <= N  do
begin
NBCOUPLES := A + B;
A := B;
B := NB COUPLES ;
I := I + 1;
end;
(* affichage des résultats *)
writeln('Au début du mois numéro ',N, ' le nombre de coules de lapins vivant dans la garenne est ',
end.
```

Remarques

1. En Turbo Pascal, un integer est codé sur 2 octets (16 bits) donc $-32768 = -2^{15} \leq integer \leq 32767 = 2^{15} - 1$.
   Si l'on prend NBCOUPLES de type integer, on obtient un message d'erreur "Stack overflow : capacité de pile dépassée" pour $N = 23$ car alors $NBCOUPLES = 46368$ alors que pour $N = 22$ on obtient $NBCOUPLES = 28657$ Turbo Pascal, un real est codé sur 6 octets (48 bits) donc $2.9 \times 10^{-39} \leq |real| \leq 1.7 \times 10^{38}$.
   Si l'on prend NBCOUPLES de type real, on obtient un message d'erreur "Stack overflow : capacité de pile dépassée" pour $N = 184$ car alors alors que pour $N = 184$ on obtient $NBCOUPLES \approx 1.2712787974 \times 10^{38}$

2.

8.4 Exemples d'algorithmes numériques

8.4.1 Euclide

PGCD

PPCM

Coefficients de Bezout

8.4.2 Horner

Calcul de $P(x_0)$, de $P^{(n)}(x_0)$

Application à la substitution

8.5 Nombres aléatoires

8.5.1 Création de $0 \leq rand() < 1$

On peut se baser sur la formule suivante $u_n =$ parrtie fractionnaire $(u_{n-1} + \pi)^5)$
Version 1 :

```
U:= ;
V := (U + pi)^5 ;
U := V - int(V) ;
print(U);
```

Comme le calcul de X^5 nécessite l'utilisation de logarithmes, on peut gagner du temps en calculant directement $X * X * X * X * X$ d'où :
Version 2 :

```
U:= ;
U :=U + pi ;
U := U*U*U*U*U;
U := U -int(U);
Print(U);
```

8.5.2 Autres nombres aléatoires

1. Jeu de dés : $Y := 1 + int(6 * rand())$ donnera un entier compris entre 1 et 6
2. Jeu de loto : $Y := 1 + int(49 * rand())$ donnera un entier compris entre 1 et 49
3. Date au 20ème siècle : $Y := 1900 + int(100 * rand())$ donnera une année comprise entre 1900 et 1999

8.5.3 Arrondi automatique au centime

$$Z := \frac{int(Z * 100 + 0.5)}{100}$$

8.6 Bibliographie

The Art of Computer Programming - Donald Knuth - Editions Addison Wesley
L'Algorithmique : votre passeport informatique pour la programmation - B. Warin - Editions Ellipses
Algorithmique numérique - C. Brezinski - Editions Ellipses
Turbo-Pascal élémentaire pour la Physique - J.Renault - Editions Dunod.
L'informatique en Sup et Spe - B.Petazoni - Editions Ellipses.
Quelques aspects du monde étrange des réels informatiques - A.Warusfel - RMS Oct 1987.
Algorithmes numériques : Analyse et mise en oeuvre
Tome 1 : Arithmétique des ordinateurs - Système linéaires
Tome 2 : Equations et Systèmes non linéaires .
J.Vignes et M.Laporte - Editions Technip