1 Cours d'algorithmique et rappels de MAPLE

1.1 Définition

Un algorithmique est un mot à l'orthographe compliquée, mais on peut définir ce mot simplement : il s'agit tout simplement d'une suite d'instructions qui, une fois exécutée conduit au résultat voulu. Par exemple, lorsque quelqu'un vous demande son chemin, vous allez lui donner une suite d'instructions à suivre pour qu'il se rende à sa destination. Bien sûr vous ne pouvez pas juste lui dire "Débrouille-toi", il s'agit là d'une instruction qu'il ne peut pas exécuter. Pour la programmation sur ordinateur, il en va de même, il ne peut comprendre que certaines instructions. Et celles-ci ne sont pas très nombreuses, donc tout va bien !

1.2 Vérifier l'algorithme

Très important : tous les programmeurs testent leurs algorithmes en se mettant à la place de la machine. C'est une démarche qu'il faut absolument avoir. Avec de l'entrainement vous le ferez de plus en plus rapidement. Cela va vous permettre de déceler la plupart des erreurs.

1.3 Les différents type d'objets dans MAPLE

- Une instruction est une phrase de MAPLE. Elle doit obligatoirement se terminer par un " ;" ou un " :". Dans ce dernier cas l'évaluation ne s'affichera pas. Une instruction renvoie une valeur, même si elle ne s'affiche pas. Par exemple : $2 : \%+1;$
- Nombres et expressions numériques : les nombres de Maple sont évalués de façon exacte, sauf si un des nombres est déjà une valeur approchée. Maple comprend alors qu'il ne sert à rien de faire du calcul exact (couteux en temps). Pour calculer une valeur approchée on utilise **evalf(expression) ;**. Essayer les instructions suivantes (tâcher de deviner à l'avance le résultat) : 2^{50}; $evalf(2)$; $2.^{50}$; $sin(2)$; $sin(evalf(2))$;
  On peut changer la précision en écrivant : $Digits := 1; 6 * 7.$;
- Chaînes alphanumériques : elles servent en général à rendre une fonction plus conviviale. On doit les écrire entre guillemets, autrement MAPLE va prendre la chaîne pour une variable, et va chercher à l'évaluer. Essayer : $"essai"[4]; cat("essai","numero",3);$
- variables : Les variables sont désignées par des chaînes alphanumériques. Maple distingue majuscules et minuscules !
- Les affectations : On peut ranger certaines valeurs (des nombres, des chaines de caractères, des expressions, des fonctions, des procédures ...) dans des boites qui ont une étiquette alphanumérique. Par exemple pour mettre 2 dans la mémoire x on écrira $x := 2;$. Si par la suite, on écrit $xx := 10^{-20} + x;$, Maple va évaluer la partie droite de l'expression, puis ranger le résultat dans la mémoire xx. Lorsqu'il aura à évaluer $10^{-20}+x$, il va commencer par évaluer chacune des deux parties avant de les aditionner. Par principe, Maple évalue toujours les arguments d'une fonction avant de l'appliquer.
  1. Que contiennent les variables x et z après ces instructions ? (surtout avant d'écrire ces instructions, deviner d'abord le résultat.
     (1.1) restart ;$z := x; x := 3; x := 2;$
     (1.2) restart ;$x := 3; z := x; x := 2;$
  2. Ecrire un algorithme permettant d'échanger les valeurs de deux mémoires. Par exemple $x := 939038697; y := 97369769;$. Bien sûr interdiction de recopier.
- Procédure : (*mieux expliquer, en particulier le remplacement des variables par exemple f :=proc(x) x :=3 ;end ; Que renvoie f(2) ?*) une procédure a la syntaxe suivante :

$$proc(var_1, var_2, \dots) inst1 \dots instn \quad end$$

  Cela sert à ranger toute une suite d'instructions que l'on veut utiliser plusieurs fois. La règle d'or en informatique c'est de ne jamais écrire deux fois la même suite d'instructions. Mieux vaut la mettre en mémoire, comme cela, on peut plus facilement changer le

programme. Par exemple $norme := proc(x,y)sqrt(x^2+y^2);end;norme(1,1);$. Maple reconnaît qu'il doit faire appelle à une procédure lorsqu'un nom de variable est suivi par des parenthèses. Pour une procédure aussi simple que dans l'exemple, c'est à dire pour une procédure qui n'utilise pas de variable intermédiaire, on préfère utiliser la syntaxe des fonctions qui est beaucoup moins lourde (voir fonction). Si on a besoin d'autres variables pour la procédure, il est plus prudent de les déclarer comme locales, cela évite bien des erreurs. Par exemple : $norme := proc(x,y)localn;n := sqrt(x^2+y^2);end;$

– fonctions : type particulier de procédure, à la syntaxe beaucoup plus simple :

$$x- > expression$$

Par exemple : $carre := x- > x^2;$, $norme := (x,y)- > sqrt(x^2+y^2);$

1. Écrire une procédure qui calcule la distance d'un point $A(x_A, y_A)$ à une droite d'équation $ax + by + c = 0$:

$$d = \frac{|ax_A + by_A + c|}{\sqrt{a^2+b^2}}$$

Utiliser **abs** pour la valeur absolue.

2. Ecrire une procédure qui, à partir de l'affixe a d'un point, de l'affixe ω du centre et de l'angle θ donne l'affixe du point image de celui d'affixe a par la rotation de centre d'affixe ω et d'angle θ.

– tableaux : on peut ranger ensemble plusieurs objets dans un tableau. La syntaxe est

$$[obj_1, obj_2, \ldots, obj_n]$$

Par exemple : $trigo := [sin, cos]; trigo[1](Pi); trigo[2](Pi);$

– Les équations : ce sont deux expressions séparées par le signe =. On confond souvent équation et affectation !

1.4 Les entrées/sorties

Dans cette catégorie d'instructions se trouvent donc les entrées (du point de vue de l'ordinateur !) qui permettent d'affecter une mémoire via un dialogue avec l'être humain situé en face, et les sorties qui permettent d'afficher les valeurs de certaines variables, ou bien des chaînes de caractères qui ont un sens (toujours pour le même humain). Lorsque, dans l'exécution d'un programme, on arrive à une instruction de cette famille, le programme peut s'arrêter et attendre que l'on fasse quelque chose. On peut trouver des programmes qui ne contiennent aucune sortie puisque par défaut, Maple affiche le dernier résultat calculé. Une bonne programmation permet d'éviter ce genre d'instructions. Mais, malgrè tout il peut être utile d'afficher certains résulats intermédiaires, notamment pour comprendre une erreur dans un algorithme. On utilise la fonction **print()**, ou bien la fonction **printf()** qui permet un affichage plus soigné. Exemple x :=3 ;print("x vaut",x) ;

1.5 Les tests

Jusqu'à présent on a fait des programmes assez peu intéressants. Il peut être utile de faire des tests dans un programme, et, suivant le résultat du test, d'exécuter une série d'instructions différentes. La syntaxe est la suivante :
If <condition>**then**<instruction 1>. . . <instruction n>**fi ;**
Les <conditions> sont soit des conditions simples : une expression - un opérateur de comparaison - une expression, soit des conditions composées de conditions simples articulées par des **and**, des **or**, des **not** et des parenthèses. Par exemple $(n\textbf{mod}2) = 0$ teste si n est congru à 0 modulo 2, autrement dit si n est pair.
Si la condition est réalisée on exécute la suite d'instructions entre **then** et **fi** sinon, le programme saute cette suite d'instructions.
On peut bien sûr imbriquer des tests et ainsi se débrouiller dans tous les cas de figure, mais il est souvent plus commode d'utiliser l'autre forme du test :

If <condition>**then**<instruction 1>. . . <instruction n>**else**<instruction n+1>. . . <instruction n+p>**fi ;**
Cette fois ci, si le test est négatif, le programme exécute les instructions $n+1 \dots n+p$, et s'il est positif les instructions $1 \dots n$

1. Écrire un programme qui donne les solutions réelles d'une équation du second degré.
2. Écrire un programme qui, étant donné la somme et le produit de deux nombres donne leurs valeurs
3. Écrire un programme qui, étant donné les équations de deux droites dans le plan, dit si elles sont parallèles, confondues ou sécante en un point.

1.6 Les boucles

On arrive enfin à la catégories d'instruction la plus difficile mais aussi la plus intéressante. On va commencer par voir la boucle la plus générale, la boucle while. La syntaxe est pratiquement la même pour les deux machines :
While <condition> **do** <instruction 1>. . . <instruction n> **od ;**
Si la condition est réalisée, on exécute les instructions $1 \dots n$, puis on retourne au début du while. En quelque sorte, tant que la condition est réalisée, on boucle sur la suite d'instructions, mais on ne peut sortir de la boucle qu'après l'instruction n.

Exemple :
nbprem :=proc(n) local i ;
$i := 1;$
while $i < 10$ **do**
if isprime(i) **then** print(i) ; fi ; $i := i + 1;$
od

Dans ce programme on va afficher les nombres premiers inférieurs à n.

1. Trouver le plus petit nombre premier supérieur à 10^200.

Il existe un autre type de boucle fréquement utilisé, il s'agit des boucles for. Celles-ci ont un compteur de boucle intégré et sont utilisées lorsque l'on veut faire un nombre fixé de fois une suite d'instructions. Dans les boucles while, il est souvent difficile de prédire le nombre de fois que l'on va boucler.