Algorithmique en seconde avec Xcas.

Bernard Parisse

Juillet 2009

Ce document présente des pistes pour le programme de seconde d'algorithmique avec Xcas. L'introduction est générale à la problématique de l'algorithmique, les paragraphes qui suivent sont spécifiques à Xcas. Il ne s'agit en aucun cas d'un manuel complet, plutot d'un tutoriel ou guide pour les professeurs qui choisiront Xcas pour enseigner l'algorithmique au lycée. Pour un manuel complet et de très nombreux exemples d'algorithmes et de programmes, on renvoie au manuel de Renée De Graeve (menu `Aide->Manuels->Programmation` de Xcas).

1 Introduction

1.1 Algorithme et langage de programmation

Comment fonctionne un programme informatique ? Il manipule des données par une suite d'instructions exécutées par le microprocesseur de l'ordinateur. Un microprocesseur actuel est capable d'exécuter plusieurs milliards d'instructions par seconde, mais ces instructions sont très simples (par exemple additionner deux nombres entiers compris entre 0 et 2^{32}), il en faut énormément pour réaliser une tache intéressante (par exemple un programme de traitement de texte contiendra quelques millions d'instructions) et leur codage est peu lisible pour un être humain. Il faut donc hiérarchiser les instructions et les données manipulées par le microprocesseur pour pouvoir exprimer la réalisation d'une tache intéressante en un nombre d'instructions contrôlable par un humain. D'où la notion de langages informatiques : on écrit dans un langage compréhensible par l'être humain un texte (le code source) qui sera traduit en une suite d'instructions compréhensible par le microprocesseur. Il existe de très nombreux langages informatiques :

- le C, et le C++ est le langage sans doute le plus utilisé,
- le Fortran très utilisé aux débuts de l'informatique dans les domaines nécessitant du calcul scientifique,
- le Cobol utilisé en gestion,
- Java, javascript, php, utilisé dans les navigateurs,
- Logo, Pascal qui étaient spécifiquement destinés à l'enseignement,
- Perl, Python, Lisp, Ada, Caml, ...

Les principales différences entre ces langages sont

- la façon de hiérarchiser les données. Certains langages controlent strictement les types de données que l'on peut manipuler (langages typés), d'autres non.

– la façon de hiérarchiser les instructions. Certains langages sont assez proches de la façon de fonctionner du microprocesseur (les langages dits impératifs), d'autres s'en éloignent plus (langages fonctionnels ou orientés objets par exemple).
– la méthode de traduction utilisée. Certains langages permettent de traduire le code source en un programme exécutable en-dehors de l'environnement de développement (on parle de langage compilé), d'autres nécessitent un environnement spécifique (soit l'environnement de développement pour les langages interprétés, soit un environnement allégé par exemple pour le langage java).

Il est important de noter que chaque langage a ses inconvénients et ses avantages, il n'y a pas de langage intrinsèquement le meilleur. Tout dépend du domaine de l'application à développer, de ce qui existe et bien sur des connaissances du développeur. Le langage de Xcas est un langage non typé, impératif (mais avec certains aspects fonctionnels) et interprété.

Comment écrit-on un programme ? Il faut d'abord se poser la question des données à manipuler : quelles sont les données du problème, comment seront-elles entrées, quelles sont les résultats, comment seront-ils transmis, quelles seront les données intermédiaires nécessaires, et comment passe-t-on des données initiales aux résultats. C'est ce qui constitue un algorithme. Par exemple pour calculer le pgcd de 2 entiers, on peut choisir de faire entrer un par un les deux entiers à l'utilisateur, puis on calcule les restes successifs des divisions euclidiennes, on s'arrête lorsque le reste est nul et on renvoie le dernier reste non nul. On peut écrire l'algorithme de manière plus formalisée :

```
entrer la valeur de a,b
tantque b est non nul faire
  r <- reste de la division de a par b
  a <- b
  b <- r
fintantque
afficher a
```

On choisit alors un langage (nous supposerons qu'il s'agit du langage de Xcas dans la suite), et on traduit l'algorithme en utilisant les mots-clefs du langage, ici

```
saisir a,b;
tantque b!=0 faire
  r := irem(a,b);
  a := b;
  b := r;
ftantque
afficher a
```

Naturellement, en fonction de la maitrise et de la complexité de la tache, on peut sauter certaines étapes et par exemple directement saisir le programme en langage Xcas qui est très proche de l'algorithme.

1.2 Pourquoi choisir Xcas ?

- le langage de Xcas est interprété et non typé, ce qui facilite la mise en oeuvre de manière interactive. Les structures de contrôle sont utilisables en français avec des délimiteurs explicites de blocs.
- Xcas est un logiciel de calcul formel que l'on peut utiliser par ailleurs en maths au lycée (pour vérifier un calcul, effectuer un calcul compliqué en se concentrant sur les notions plutôt que sur la mécanique de calcul ou tout simplement explorer). Or utiliser un logiciel de calcul formel nécessite de savoir écrire une ligne de commande ce qui est très proche de l'écriture d'instructions en algorithmique, on peut passer progressivement de l'un à l'autre (c'est d'ailleurs ainsi que ce document est conçu).
- Xcas possède de nombreuses instructions géométriques, toutes utilisables en ligne de commande ou dans un programme, permettant de traiter de manière algorithmique des situations rencontrées en géométrie.
- Xcas permet l'exécution en mode pas à pas, ce qui peut faciliter la compréhension d'algorithmes montrés par le professeur mais aussi la mise au point d'un programme par l'élève.
- Xcas permet aussi de faire de la géométrie dynamique dans le plan, dans l'espace, et propose un tableur. On peut donc faire toutes les maths au lycée et au-delà avec le même logiciel, ce qui rentabilise le temps passé à le maitriser.

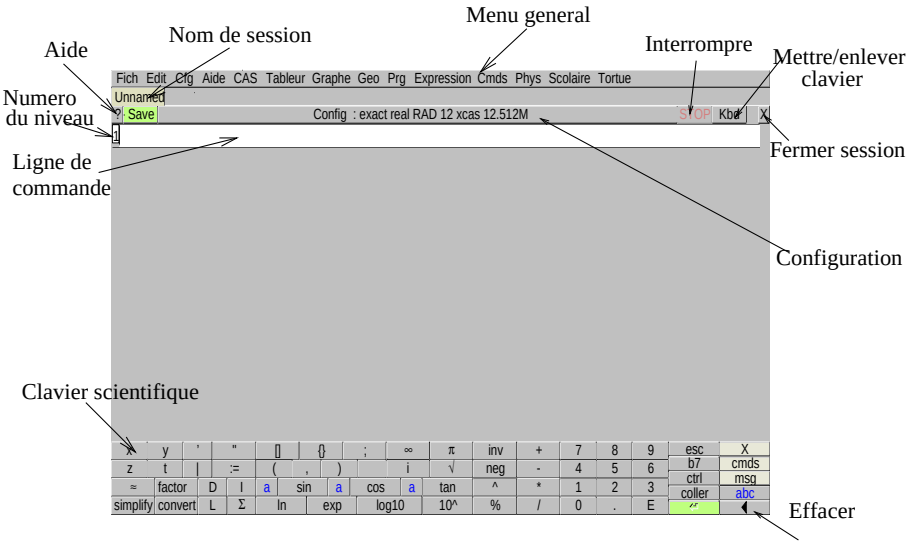
2 Premiers pas avec Xcas

Pour télécharger Xcas, allez sur le site
`http://www-fourier.ujf-grenoble.fr/~parisse/install_fr.html`
Pour traiter les exemples, il est conseillé d'ouvrir Xcas :

- Sous Windows en installation locale, on clique sur l'icone `xcasfr` du bureau.
- Sous Linux avec Gnome, on clique sur Xcas du menu Education. Sinon, ouvrir un terminal et taper `xcas &`.
- sur Mac, cliquez sur Xcas dans le menu Applications du Finder.

Lors de la première utilisation, choisissez `Xcas` lorsqu'on vous demande de choisir une syntaxe (sauf si vous connaissez le langage Maple). Nous donnons ici seulement le minimum de l'interface à connaitre pour commencer à programmer. On consultera plutôt le manuel Débuter en calcul formel ou les autres manuels (menu Aide) pour apprendre à utiliser les fonctionnalités de Xcas en calcul formel, géométrie, tableur, etc..

L'interface (vous pouvez la redimensionner) apparaît comme suit au lancement de `Xcas` :



De haut en bas cette interface fait apparaître :

- un barre de menu gris cliquable : `Fich`, `Edit`, `Cfg`, `Aide`, `CAS`, `Tableur`, `Graphe`, `Geo`, `Prg`,...
- un onglet indiquant le nom de la session, ou `Unnamed` si la session n'a pas encore été sauvegardée,
- une zone de gestion de la session avec :
  - un bouton `?` pour ouvrir l'index de l'aide
  - un bouton `Save` pour sauvegarder la session,
  - un bouton `Config: exact real ...` affichant la configuration et permettant de la modifier,
  - un bouton `STOP` permettant d'interrompre un calcul trop long,
  - un bouton `Kbd` pour faire apparaitre un clavier ressemblant à celui d'une calculatrice, qui peut faciliter vos saisies, c'est saisir(a)
  - un bouton `X` pour fermer la session
- une zone rectangulaire blanche numérotée 1 (première ligne de commande) dans laquelle vous pouvez taper votre première commande (cliquez si nécessaire pour faire apparaitre le curseur dans cette ligne de commande) : `1+1`, suivi de la touche "Entrée" ("Enter" ou "Return" selon les claviers). Le résultat apparaît au-dessous, et une nouvelle ligne de commande s'ouvre, numérotée 2.

Vous pouvez modifier l'aspect de l'interface et sauvegarder vos modifications pour les utilisations futures (menu `Cfg`).

Vous n'avez pour l'instant qu'à entrer des commandes dans les lignes de commandes successives. Si vous utilisez la version html de ce cours, vous pouvez copier-coller les commandes proposées depuis votre navigateur. Chaque ligne de commande saisie est exécutée par la touche "Entrée". Essayez par exemple d'exécuter les commandes suivantes.

```
1/3+1/4
```

```
sqrt(2)^5
resoudre(x+3=1,x)
50!
```

Toutes les commandes sont gardées en mémoire. Vous pouvez donc remonter dans l'historique de votre session pour modifier des commandes antérieures. Essayez par exemple de changer les commandes précédentes en :

```
1/3+3/4
sqrt(5)^2
resoudre(2*x+3=0)
500!
```

Notez que

- pour effacer une ligne de commande, on clique dans le numéro de niveau à gauche de la ligne de commande, qui apparait alors en surbrillance. On appuie ensuite sur la touche d'effacement. On peut aussi déplacer une ligne de commande avec la souris.
- Le menu `Edit` vous permet de préparer des sessions plus élaborées qu'une simple succession de commandes. Vous pouvez créer des groupes de lignes de commandes (sections), ajouter des commentaires ou fusionner des niveaux en un seul niveau.
- Le menu `Prg` contient la plupart des instructions utiles pour programmer et permet d'ouvrir un éditeur de programmes.

3 Données, variables, affectation

Pour stocker une donnée en mémoire, on utilise une variable. Une variable possède un nom, qui est composé d'au moins un caractère alphabétique (de A à Z ou de a à z) suivi ou non d'autres caractères alphanumériques, par exemple `a`, `var`, `b12`. Xcas est sensible à la différence entre majuscules et minuscules. Une variable peut contenir n'importe quel type de donnée.

Pour donner une valeur à une variable

- on écrit un nom de **variable**, suivi par l'instruction d'**affectation** `:=` suivi par la valeur. Par exemple `a := 1.2`
- ou on utilise l'instruction `saisir` suivi du nom de variable que l'on met entre parenthèses. Par exemple `saisir(a)` ou encore `saisir(a,b)` ou encore pour faciliter la saisie `saisir("Entrez votre nom",a,"Entrez votre age",b)`.

Pour utiliser la valeur d'une variable

- on tape une ligne contenant le nom de la variable et la touche Entrée, la variable sera automatiquement remplacée par sa valeur. Par exemple si on tape `a:=2` puis `a+1` on obtient 3.
- on peut aussi afficher la valeur d'une variable au cours de l'exécution d'un programme en utilisant la commande `afficher`. Cela permet d'afficher un résultat intermédiaire dans une zone située avant la ligne contenant le résultat du programme. Par exemple `afficher(a+1)` affichera `3` en bleu dans cette zone.

Xcas peut manipuler différents types de données dont :

– les entiers, les fractions, les nombres flottants. La différenciation se fait pour les nombres par la présence d'un point séparateur décimal par exemple `1`, `2/3`, `1.1` (on peut aussi utiliser la notation scientifique mantisse, exposant comme dans `1e-7`)
– les paramètres formels, par exemple `x`, `t`, `z`,
– les expressions, par exemple `x+1`, `x^2+y^2`,
– les fonctions, par exemple `f:=x->x^2` ou `f(x):=x^2`,
– les chaines de caractères, par exemple `s:="bonjour"`. On accède à un caractère d'une chaine en donnant le nom de la chaine indicié par un entier compris entre 0 et la taille de la chaine moins 1 (même convention qu'en langage C), par exemple `s[0]` renvoie `"b"`.
– les listes et les vecteurs, par exemple `a:=[1,2,3]`. On accède à un élément d'une liste en donnant le nom de la liste indicié par un entier compris entre 0 et la taille de la liste moins 1 (même convention qu'en langage C), par exemple `a[0]` renvoie 1. On peut créer des listes de taille donnée avec une formule de remplissage (par exemple `l:=[0$10]`, `l:=[j^2$j=1..10]`, voir aussi `seq`, `makelist`, `ranm`)
– des objets géométriques ou plus généralement graphiques (point, droite, cercle, polygone, plan, sphère, courbe représentative d'une fonction, etc.) obtenus par une instruction graphique (cf. section 4.2)

Exercices :

– Ecrire dans une ligne de commande un programme de conversion d'euros en francs (on demande avec `saisir` le montant en euros, on convertir en francs et on affiche le résultat avec `afficher`)
– Ecrire un programme de résolution de l'équation $ax + b = 0$ (on entre a et b et le programme renvoie la valeur de x ou affiche un message d'erreur)

4 Instructions

4.1 Syntaxe

Une instruction valide dans Xcas doit suivre une syntaxe précise qui ressemble à l'écriture d'une expression en mathématique. Les différents éléments d'une instruction simple peuvent être des nombres (entiers, réels, flottants), des noms de variables, des opérateurs (par exemple `+`), des appels à des fonctions (l'argument ou les arguments se trouvant entre les parenthèses séparés par une virgule s'il y a plusieurs arguments) : par exemple `sqrt(3)` pour désigner la racine carrée de 3 ou `irem(26,3)` pour désigner le réste de la division euclidienne de 26 par 3).

Il faut prendre garde à la priorité entre les différentes opérations. Par exemple dans `1+2*3` l'opération `*` est effectué avant `+` comme en mathématiques. On peut utiliser des parenthèses pour forcer l'ordre des opérations, par exemple `(1+2)*3`.

Exercice : Calculer
`5/2/3`, `5/(2/3)`, `5/2*3`, `5/(2*3)`, `5^2*3`, `5^(2*3)`,`5*2^3`, `(5*2)^3`
Conclure sur les priorités relatives de la multiplication, de la division et de la puissance.

Attention, la multiplication doit toujours être indiquée explicitement contrairement aux mathématiques. Ainsi l'écriture `ab` ne désigne pas le produit de 2 variables `a` et `b` mais une variable dont le nom est `ab`.

Lorsqu'on veut exécuter plusieurs instructions à la suite, on termine chaque instruction par le caractère `;` (ou par les caractères `:;` si on ne souhaite pas afficher le résultat de l'instruction).

4.2 Instructions graphiques

Toutes les instructions du menu `Geo` ont un résultat graphique (à l'exception des instructions du menu Mesure), par exemple :

– `point(1,2)` affiche le point de coordonnées 1 et 2,

– `droite(A,B)` la droite passant par deux points A et B définis auparavant.

Lorsqu'une ligne de commande contient une instruction graphique, le résultat est affiché dans un repère 2-d ou 3-d selon la nature de l'objet généré. On peut controler le repère avec les boutons situés à droite du graphique, par exemple orthonormaliser avec le bouton $\perp$. Si une ligne de commande contient des instructions graphiques et non graphiques, c'est la nature de la dernière instruction qui décide du type d'affichage. On peut donner des attributs graphiques aux objets graphiques en ajoutant à la fin de l'instruction graphique l'argument `affichage=...`, argument dont la saisie est facilitée par le menu `Graphic->Attributs`.

L'instruction `A:=point(click())` permet de définir une variable contenant un point du plan que l'on clique avec la souris. C'est un peu l'analogue géométrique de l'instruction `saisir`.

Lorsqu'on veut voir sur un même graphique le résultat de plusieurs lignes de commandes, on crée une figure avec le menu `Geo->Nouvelle figure->graph geo2d`, la figure correspond alors aux lignes de commande situées à gauche. Dans ce mode, on peut créer les objets graphiques en ligne de commande ou pour certains directement à la souris en sélectionnant un `Mode` et en cliquant. On peut aussi déplacer un point en mode `Pointeur` et observer les modifications des objets géométriques qui en dépendent.

Exemples :

– calcul du milieu de 2 points sans utiliser l'instruction `milieu`.

```
A:=point(click()); B:=point(click());
xm:=(abscisse(A)+abscisse(B))/2;
ym:=(ordonnee(A)+ordonnee(B))/2;
C:=point(xm,ym)
```

– droite verticale passant par un point

```
A:=point(click()); D:=droite(x=abscisse(A))
```

Exercices :

– faire cliquer un point puis afficher son symétrique par rapport à l'axe des x.

– Faire cliquer deux points A et B à la souris, déterminer un 3ème point C tel que ABC soit rectangle en A avec AC de longueur 1.

– Créer une figure (menu Geo, nouvelle figure, graph geo2d). Créer 3 points (soit avec l'instruction `point`, soit en passant en mode `point` et en cliquant 3 points

à la souris). Afficher le centre du cercle circonscrit du triangle formé par ces 3 points en utilisant uniquement les instruction `mediatrice` et `inter_unique`.

4.3 Tests

On peut tester l'égalité de 2 expressions en utilisant l'instruction `==`, alors que `!=` teste si 2 expressions ne sont pas égales. On peut aussi tester l'ordre entre 2 expressions avec `<`, `<=`, `>`, `>=`, il s'agit de l'ordre habituel sur les réels pour des données numériques ou de l'ordre lexicographique pour les chaines de caractères.

Un test renvoie 1 s'il est vrai, 0 s'il est faux. On peut combiner le résultat de deux tests au moyen des opérateurs logiques `&&` (et logique), `||` (ou logique) et on peut calculer la négation logique d'un résultat de test avec `!` (négation logique). On utilise ensuite souvent la valeur du test pour exécuter une instruction conditionnelle comme l'instruction `si` :

```
si condition alors bloc_vrai sinon bloc_faux fsi
```

ou pour arrêter une boucle en cours d'exécution.

Par exemple, on pourrait stocker la valeur absolue d'un réel x dans y par :

```
si x>0 alors y:=x; sinon y:=-x; fsi;
```

(on peut bien sur utiliser directement `y:=abs(x)`).

Remarque : Xcas admet aussi une syntaxe compatible avec le langage C :

```
if (condition) { bloc_vrai } else { bloc_faux }.
```

Exercices :

- Saisir deux nombres réels et tester s'ils sont de même signe.
- Créer une figure (menu Geo->Nouvelle figure->graph, geo 2d) puis la `droite` d'équation $y = 2x + 1$, puis le `point` O origine, puis un point A quelconque, faire afficher si A est du même côté de la droite que O. De même pour O un point quelconque. Faire bouger le point A à la souris (en passant en `Mode` pointeur) pour vérifier les différentes possibilités.
- Dans un brin d'ADN, les caractères A et T d'une part et C et G d'autre part sont dits complémentaires. Saisir deux caractères, tester s'ils sont complémentaires.
- Tester si un triangle dont on fait cliquer les 3 sommets à l'utilisateur est rectangle (on pourra tester le théorème de Pythagore en chacun des trois sommets, en utilisant l'instruction `distance` pour avoir la distance entre 2 sommets).

4.4 Boucles

On peut exécuter des instructions plusieurs fois de suite en utilisant une boucle définie (le nombre d'exécutions est fixé au début) ou indéfinie (le nombre d'exécutions n'est pas connu). On utilise en général une variable de controle (indice de boucle ou variable de terminaison).

- Boucle définie

  ```
  pour ... de ... jusque ... faire ... fpour
  ```

  Exemple, calcul de 10 !

  ```
  f:=1; pour j de 1 jusque 10 faire f:=f*j; fpour;
  ```

On peut ajouter un pas s'il est diffÃl'rent de 1, par exemple le produit des nombres pairs de 2 à 10

```
f:=1; pour j de 2 jusque 10 pas 2 faire f:=f*j; fpour;
```

Attention, vous ne pouvez pas utiliser `i` comme indice de boucle, car `i` est prédéfini (comme $\sqrt{-1}$).

- Boucle indéfinie
  - `repeter ... jusqu_a ...`
    Exemple, saisir un nombre entre 1 et 10
    ```
    repeter saisir("Entrer un nombre entre 1 et 10",a); jusqu_a a>=1 && a<=10;
    ```
  - `tantque ... faire ... ftantque`
    Exemple, algorithme d'Euclide
    ```
    tantque b!=0 faire r:=irem(a,b); a:=b; b:=r; ftantque
    ```

Remarques :

- Xcas accepte aussi la syntaxe de type C
  `for(init;condition;incrementation){ instructions }`
  l'arrêt de boucle en cours d'exécution (`if (...) break;`) dont l'usage peut éviter l'utilisation de variables de contrôle compliquées, et le passage immédiat à l'itération suivante (mot-clef `continue`) qui évite des forêts d'`if`.
- Lorsqu'on crée des objets graphiques dans une boucle, seul le dernier est affiché (car c'est le résultat de l'évaluation de la boucle). Pour afficher tous les objets graphiques d'une boucle, il faut les conserver dans une variable au cours de la boucle, en pratique on initialise une séquence `res:=NULL` avant la boucle, et on écrit dans la boucle `res:=res,objet_graphique`. Notez que les objets graphiques intermédiaires sont de toutes facons affichés dans la fenêtre `DispG` (menu `Cfg->Montrer`).

Exemples :

- calculer la somme des cubes des nombres de 1 à 10.
- Tester que 2 chaines représentant des brins d'ADN sont complémentaires (A correspond à T et C à G), en utilisant les instructions suivantes taille d'une chaine `s size(s)`, premier caractére `s[0]`, caractère suivant `s[1]`, ..., dernier caractère `s[size(s)-1]`. On pourra aussi traduire une chaine en la chaine complémentaire, ou en la chaine d'ARN correspondant (A devient U, T devient A, C devient G, G devient C).
- afficher la liste des diviseurs de 100 en testant si les entiers de 1 à 100 divisent 100
- afficher quelques points d'un graphe de fonction (par exemple en calculant les images des nombres de -2 à 2 avec un pas de 0.1, pour la fonction $x \to x^2$)
- Construire un polygone régulier à 10 côtés dont le centre est donné, ainsi qu'un sommet (on pourra utiliser l'instruction `rotation`).

5 Fonctions

Pour réaliser des taches un peu plus complexes, il devient intéressant de les subdiviser en plusieurs entités indépendantes appelées fonctions, qui permettent d'étendre les possibilités des instructions de Xcas. Une fonction peut avoir 0, 1 ou plusieurs ar-

guments (comme la fonction racine carrée `sqrt()` en a un). Elle calcule une valeur, appelée valeur de retour.

La définition de fonctions peut parfois se faire directement avec une formule (fonction définie algébriquement) mais nécessite parfois un algorithme plus complexe. C'est peut-être à cet endroit qu'il faut s'arrêter en classe de seconde (au moins en terme d'exigences), en réservant les fonctions plus complexes (et les variables locales) à la première scientifique et les fonctions récursives à la classe de terminale, conjointement à l'étude de la récurrence.

5.1 Fonctions simples

Il s'agit de fonctions définies par une formule algébrique. Exemples :

- pour définir $f(x) = x^2 + 1$, on tape `f(x):=x^2+1`
- pour définir la valeur absolue sans utiliser `abs`, on tape :

```
absolu(x):= si x<0 alors -x sinon x fsi;
```

Exercices :

- définir une fonction par morceaux.
- définir une fonction `max2` calculant le maximum de 2 entiers (sans utiliser `max`) puis de 3 entiers en appelant la fonction précédente ou sans appeler la fonction précédente. Observer l'intérêt d'utiliser la fonction `max2`.
- définir une fonction renvoyant le caractère complémentaire dans un brin d'ADN

Attention

Il faut faire la différence entre fonction et expression. Par exemple `a:=x^2-1` définit une expression alors que `b(x):=x^2-1` définit une fonction. On peut donc écrire `b(2)` qui renverra 4 mais l'analogue avec a serait `subst(a,x=2)`. On peut écrire `factor(a)` mais l'analogue avec b sera `factor(b(x))`. On peut composer des fonctions, par exemple la fonction $c = b \circ b$ est `c:=b@b` et aussi construire une fonction à partir d'une expression avec l'instruction `unapply` par exemple la fonction `b` définit par l'expression `a` est `b:=unapply(a,x)`.

5.2 Fonctions plus complexes

La plupart des fonctions ne peuvent pas être définies simplement par une formule algébrique. On doit souvent calculer des données intermédiaires, faire des tests et des boucles. Il faut alors définir la fonction par une suite d'instructions, délimitées par `{ ... }`.

La valeur calculée par la fonction est alors la valeur calculée par la dernière instruction ou peut être explicitée en utilisant le mot-clef `return` suivi de la valeur à renvoyer, par exemple si une fonction calcule plusieurs valeurs on peut les renvoyer dans une liste ou une séquence. Notez que l'exécution de `return` met un terme à la fonction même s'il y a encore des instructions après).

Pour éviter que les données intermédiaires n'interfèrent avec les variables de la session principale, on utilise un type spécial de variables, les variables locales, dont la valeur ne peut être modifiée ou accédée qu'à l'intérieur de la fonction. On utilise à cet effet le mot-clef `local` suivi par les noms des variables locales séparés par des virgules.

Comme le texte définissant une telle fonction ne tient en général pas sur une ou deux lignes, il est commode d'utiliser un éditeur de programmes pour définir une fonction. Pour cela, on utilise le menu `Prg->Nouveau programme` de Xcas. Le menu `Prg->Ajouter` facilite aussi la saisie des principales commandes de programmation.

Exemple : encore le PGCD mais sous forme de fonction

```
pgcd(a,b):={
  local r;
  tantque b!= faire
    r:=irem(a,b);
    a:=b;
    b:=r;
  ftantque
  return a;
}
```

On clique ensuite sur le bouton OK, si tout va bien, le programme `pgcd` est défini et on peut le tester dans une ligne de commande par exemple par `pgcd(25,15)`.

Dans la section 5.4, on verra comment exécuter en mode pas à pas un programme, ce qui peut servir à comprendre le déroulement d'un algorithme, mais aussi à corriger un programme erroné.

Exercices :

- simplifier la fonction de calcul du caractère complémentaire de l'ADN en utilisant `return` pour éviter les tests imbriqués. Puis utiliser cette fonction pour créer une fonction renvoyant la chaine d'ADN complémentaire ou la chaine d'ARN.
- Reprendre les énoncés des exercices précédents en créant des fonctions et en déclarant en variables locales les variables qui servent à effectuer des calculs intermédiaires.

À ce niveau, on peut commencer à proposer des petits projets. Par exemple pour des joueurs de tarot, écrire un programme pour tenir le décompte des points. Pour les personnes intéressées par l'ADN, on peut traduire une chaine d'ADN en la protéine qu'elle code, en tenant compte des codons de start et de stop (triplets de bases d'ARN indiquant le début du codage et la fin du codage de la protéine), en testant également la présence de "sites promoteurs".

	U			C			A			G			
U	UUU	Phe	[F]	UCU	Ser	[S]	UAU	Tyr	[Y]	UGU	Cys	[C]	U
	UUC	Phe	[F]	UCC	Ser	[S]	UAC	Tyr	[Y]	UGC	Cys	[C]	C
	UUA	Leu	[L]	UCA	Ser	[S]	UAA	*STOP*		UGA	*STOP*		A
	UUG	Leu	[L]	UCG	Ser	[S]	UAG	*STOP*		UGG	Trp	[W]	G
C	CUU	Leu	[L]	CCU	Pro	[P]	CAU	His	[H]	CGU	Arg	[R]	U
	CUC	Leu	[L]	CCC	Pro	[P]	CAC	His	[H]	CGC	Arg	[R]	C
	CUA	Leu	[L]	CCA	Pro	[P]	CAA	Gln	[Q]	CGA	Arg	[R]	A
	CUG	Leu	[L]	CCG	Pro	[P]	CAG	Gln	[Q]	CGG	Arg	[R]	G
A	AUU	Ile	[I]	ACU	Thr	[T]	AAU	Asn	[N]	AGU	Ser	[S]	U
	AUC	Ile	[I]	ACC	Thr	[T]	AAC	Asn	[N]	AGC	Ser	[S]	C
	AUA	Ile	[I]	ACA	Thr	[T]	AAA	Lys	[K]	AGA	Arg	[R]	A
	AUG	Met	[M]	ACG	Thr	[T]	AAG	Lys	[K]	AGG	Arg	[R]	G
G	GUU	Val	[V]	GCU	Ala	[A]	GAU	Asp	[D]	GGU	Gly	[G]	U
	GUC	Val	[V]	GCC	Ala	[A]	GAC	Asp	[D]	GGC	Gly	[G]	C
	GUA	Val	[V]	GCA	Ala	[A]	GAA	Glu	[E]	GGA	Gly	[G]	A
	GUG	Val	[V]	GCG	Ala	[A]	GAG	Glu	[E]	GGG	Gly	[G]	G

Avec des élèves motivés (plutôt en 1ère ou en terminale...), on peut peut-être aller plus loin, par exemple écrire une fonction de score pour étudier la proximité de deux chaines

d'ADN, cf. par exemple
`www-fourier.ujf-grenoble.fr/~parisse/info/dynamic/dynamic.html`

5.3 Récursivité

On peut dans une fonction faire appel à cette fonction avec un argument "plus simple". Par exemple, l'algorithme d'Euclide peut s'énoncer sous la forme

Le PGCD de a et b est a si b est nul et est le PGCD de b et du reste de la division euclidienne de a par b sinon.

On peut le traduire en Xcas par

```
pgcdr(a,b):=si b==0 alors a sinon pgcdr(b,irem(a,b)); fsi;
```

Dans certains cas, la récursivité permet de simplifier grandement la conception d'un algorithme, par exemple pour le calcul du reste de a^n par un entier fixé m, en distinguant n pair et n impair et en utilisant pour $n > 0$ pair :

$$a^n \pmod m = (a^{n/2} \pmod m)^2 \pmod m$$

et pour $n > 1$ impair :

$$a^n \pmod m = (a \times (a^{(n-1)/2} \pmod m)^2) \pmod m$$

Attention à bien s'assurer que le programme récursif se termine bien. En particulier quand on teste un programme récursif, il est conseillé de commencer par tester le cas où la récursion doit se terminer. Notez que Xcas limite par défaut à 25 le nombre de récursions.

Attention aussi, un algorithme récursif peut être très inefficace, par exemple si on calcule les termes de la suite de Fibonacci par la formule

```
F(n):=si n<2 alors 1; sinon F(n-1)+F(n-2); fsi;
```

alors le calcul de F_5 nécessite le calcul de F_4 et F_3 mais le calcul de F_4 demande lui-même le calcul de F_3 et F_2, on calcule donc deux fois F_3, trois fois F_2, cinq fois F_1. Il est plus efficace dans ce cas d'écrire une boucle.

5.4 Exécuter en pas à pas et mettre au point

La commande `debug` permet de lancer un programme en mode d'exécution pas à pas. Elle ouvre une fenêtre permettant de diriger l'exécution du programme passé en argument. Par exemple, on entre le programme suivant (calcul de somme de carrés de 1 à n) :

```
carres(n):={
  local j,k;
  k:=0;
  for (j:=1;j<n+1;j++) {
    k:=k+j^2;
  }
  return k;
}:;
```

On tape pour debugger le programme `carres` ci-dessus :

```
debug(carres(5))
```

cela ouvre la fenêtre du debugger. En cliquant sur le bouton `sst` on peut exécuter pas à pas le programme en visualisant l'évolution des valeurs des variables locales et des paramètres du programme. Cela permet de détecter la grande majorité des erreurs qui font qu'un programme ne fait pas ce que l'on souhaite. Pour des programmes plus longs, le debugger permet de contrôler assez finement l'exécution du programme en placant par exemple des points d'arrêt.

Exercices :

- Exécuter en mode pas à pas le programme `pgcd` pour quelques valeurs des arguments.
- Écrire le programme `cube` qui calcule

$$\sum_{j=1}^{j=n} j^3$$

  Puis,exécuter en mode pas à pas le programme `cube` pour quelques valeurs de l'argument n.

6 Référence

Données		
Type	Exemple	Instructions relatives
Entier	1	
Réel	1.2	
Fraction	1/3	
Complexe exact	1+2*i	
Complexe approché	1.2+2.3*i	
Variable	x	`:=, purge, assume saisir, afficher`
Expression	1+2*sin(x)	`part`
Liste	{1, 2}	L[1] (indice 0 à taille-1)
Vecteur	[1, 2]	v[0] (indice 0 à taille-1)
Matrice	[[1,2],[3,4]]	v[1,0] (indice 0 à taille-1)
Chaine	`"bonjour"`	s[2] (indice 0 à taille-1)

Constantes prédéfinies	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i = \sqrt{-1}$
`infinity`	∞
`+infinity` ou `inf`	$+\infty$
`-infinity` ou `-inf`	$-\infty$

Opérateurs logiques			
==	$=$	!=	non
<	$<$	>	$>$
<=	$\leq$	>=	$\geq$
&&	et	||	ou

Structures de controle :
- `si` condition `alors` bloc vrai `sinon` bloc faux `fsi`
- `if (` condition `){` bloc vrai `} else {` bloc faux `}`
- `pour` variable `de` init `jusque` final [`pas` ...] `faire` ... `fpour`
- `tantque` condition `faire` corps de boucle `ftantque`
- `for(`initialisation `;`test `;`incrementation`){` corps de boucle `}`

7 Exemples de corrections

- Données, variables
  - `saisir(a); afficher(a*6.5,"francs");`
  - `saisir(a,b); afficher("x=",-b/a);`
- Instructions graphiques
  - `A:=point(click()); symetrie(droite(y=0),A);`
  - ```
    A:=point(click()); B:=point(click());C:=point(click());
    m1:=mediatrice(A,B); m2:=mediatrice(B,C);
    O:=inter_unique(m1,m2); cercle(O,distance(O,C));
    ```
  - ```
    A:=point(click()); B:=point(click());
    inter(cercle(A,1),perpendiculaire(A,droite(A,B)))
    ```
- Tests
  - `saisir(x,y); si x*y>=0 alors afficher("meme signe"); sinon afficher("signe`
  - `si 2*abscisse(A)+1-ordonnee(A)>=0 alors afficher("du meme cote"); sinon af`
  - ```
    saisir(c1,c2);
    si (c1=="A" et c2=="T") ou (c1=="T" et c2=="A")
       ou (c1=="C" et c2=="G") ou (c1=="G" et c2=="C")
    alors "complementaires" sinon "non complementaires" fsi;
    ```
  - ```
    A:=point(click()); B:=point(click());C:=point(click());
    AB2:=distance2(A,B); AC2:=distance2(A,C); BC2:=distance2(B,C)
    si AB2==AC2+BC2 alors afficher("rectangle en C"); fsi;
    si AC2==AB2+BC2 alors afficher("rectangle en B"); fsi;
    si BC2==AB2+AC2 alors afficher("rectangle en A"); fsi;
    ```

- Boucles
  - `s:=0; pour j de 1 jusque 10 faire s:=s+j^3; fsi;`
  - ```
    pour j de 1 jusque 100 faire
     si irem(100,j)=0 alors afficher(j); fsi;
    fpour
    ```
  - ```
    res:=NULL;
    pour j de -2 jusque 2 pas 0.1 faire
    ```

```
   res:=res,point(j,j^2);
  fpour;
```
–
```
  A:=point(click()); B:=point(click()); res:=B;
  pour j de 1 jusque 10 faire
   res:=res,rotation(A,2*pi/10,res[j-1]);
  fpour;
  polygone(res);
```
– Fonction simple définie par morceaux, par exemple

$f(x) = x^2 - 1$ si $x > 1$, $\quad f(x) = 0$ si $x \in [-1, 1]$, $\quad f(x) = x + 1$ si $x < -1$

```
f(x):= si x>1 alors x^2-1;
 sinon si x<-1 alors x+1; sinon 0; fsi;
fsi;
```