Programmation

Alain Busser

Juin 2009

Contents

Introduction

Écrire un programme se fait toujours dans un ***langage*** de programmation. Mais le programme a vocation à être ***exécuté***, ce qui n'est possible que s'il est écrit dans un langage compréhensible par l'ordinateur. La traduction est faite par

- Un ***compilateur***, qui à partir du code écrit dans le langage de programmation, crée un exécutable, une fois pour toutes. La plupart des logiciels du commerce sont fabriqués comme ça, le plus souvent en langage C.

- Un ***interpréteur***, qui examine, traduit et exécute chaque ligne à la fois. C'est le meilleur moyen pour tester un programme en cours de création puisqu'on peut savoir où est apparu le problème qui bloque tout...
  Les langages interprétés tendent à être plus lents que les langages compilés, mais plus souples notamment en phase d'apprentissage.

La phase essentielle de la conception d'un programme est donc le ***déboguage***, ou "debuggage", qui consiste à essayer de savoir pourquoi le programme ne fait pas ce qu'il devrait faire. Le plus souvent les erreurs sont de ***syntaxe***, comme l'oubli d'une virgule. Un environnement de développement peut être jugé meilleur qu'un autre, simplement parce qu'il analyse efficacement les erreurs de syntaxe.
Le programme est tapé au clavier sous un logiciel spécialement prévu pour, appelé ***éditeur de texte***. Un bon éditeur de texte peut considérablement faciliter la tâche du programmeur. Ainsi un développeur de programmes pour calculatrices graphiques peut très bien vouloir écrire et tester son programme sur ordinateur avant de le transférer sur la calculatrice, question de confort...
Un bon éditeur de texte possède les fonctions suivantes:

- L'indentation automatique: Les lignes à l'intérieur des boucles et des tests doivent être en recul pour qu'on voie mieux les structures (boucles, tests, sous-programmes).

- La coloration syntaxique: Que les mots clés comme "for" ou "if" apparaissent dans une couleur différente, aide l'œil à les repérer. Idem pour les nombres par exemples qui doivent être dans une couleur différente.

- Un menu en général en haut de l'écran, qui permet de choisir les mots-clés dans une liste, au lieu d'avoir à les taper: Si on ne se rappelle pas comment exactement on écrit le nombre π par exemple, on cherche celui-ci dans le menu pour avoir par exemple "Math.PI".

- Éventuellement, une numérotation des lignes; pratique lorsque le compilateur annonce une erreur de syntaxe à la ligne 637...

Quelques éditeurs de texte plus ou moins intéressants:

- ***WordPad*** sous Windows, simple et léger mais peu puissant

- JE, en java, très léger, et spécialisé dans la programmation.

- Geany, Vim etc. sous Linux (il y en a plein sous Linux...)

- ***Jext*** est en java, multiplateforme et francophone...

- Toujours en java, ***netbeans*** est extrêmement puissant.

- ***emacs*** est efficace pour ce qui est d'intégrer les compilateurs...

- la version windows du logiciel POVray est fournie avec un éditeur de texte qui frôle la perfection...
  Pour un exemple de ce qu'apporte un éditeur de texte, voici comment ***gedit*** (l'éditeur de *Gnome*) affiche le source C ci-dessous:

```c
#include <stdio.h>
#include <math.h>

void main()
{
        float r=1,t;
        float s=2/r;
        while (fabs(r-s)>=1e-8)
        {
                t=(r+s)/2;
                r=t;
                s=2/r;
        }
        printf("%1.8f \n" , r);
}
```

L'activité de base du programme sera sûrement l'***affectation*** d'une valeur à une variable, comme par exemple si on veut que x soit égal à 2 (pour d'obscures raisons), on devrait écrire quelque chose comme $2 \rightarrow x$, ou, plus concrètement, $x \leftarrow 2$. Or le clavier d'un ordinateur ne possède pas de moyen simple de taper une flèche, et on a vite codé la flèche vers la gauche par "=": $x = 2$ au lieu de $x \leftarrow 2$. Si on veut incrémenter x, on écrira alors $x = x + 1$ au lieu de $x \leftarrow x + 1$, ce qui est un peu choquant pour un mathématicien ! Alors on a proposé la notation $x := 2$ qui est plus acceptable mais pas très lisible. Pire: On a parfois besoin d'une vraie égalité comme dans "for i=1 to 10" ou dans les tests "if x=0"; alors depuis le langage C, la vraie égalité est notée "==" pour la distinguer de l'affectation qui elle est notée "=". On se demande parfois si les inventeurs de langages de programmation ne font pas exprès de compliquer les choses. Par exemple, on peut taper une flèche au clavier en entrant "$<-$" ou "$<--$" mais il semble que ça ait été peu fait...

Exemple étudié

Il faudrait sans doute être un Babylonien de l'antiquité pour comprendre le sens de cette tablette:

mais il paraît qu'elle décrit une méthode pour calculer une valeur approchée rationnelle de $\sqrt{2}$:

- Si r est une valeur approchée par défaut de $\sqrt{2}$, alors $r < \sqrt{2}$, d'où $\frac{2}{r} > \frac{2}{\sqrt{2}} = \sqrt{2}$, et $\frac{2}{r}$ est une valeur approchée par excès de $\sqrt{2}$.
- Réciproquement, si r est une valeur approchée par excès de $\sqrt{2}$, alors $\frac{2}{r}$ en est une valeur approchée par défaut.

- On prend alors leur moyenne arithmétique en espérant qu'elle tombera plus près de $\sqrt{2}$
- On peut éventuellement recommencer en remplaçant r par la nouvelle valeur approchée $\frac{r+\frac{2}{r}}{2}$

On constate qu'on calcule les termes successifs d'une suite définie par récurrence, mais le vocabulaire des suites n'est pas nécessaire pour décrire l'algorithme suivant[1]:

- On choisit un nombre de départ r; dans l'exemple ce sera 1;
- on calcule $\frac{r+\frac{2}{r}}{2}$;
- on compare ce dernier avec r;
- si la différence est suffisamment petite (mettons inférieure à 10^{-8}), on estime qu'on a une valeur approchée de $\sqrt{2}$ à 10^{-8} près, et on a fini.
- sinon on recommence avec la nouvelle valeur de r

Cet exemple a été choisi parce qu'il utilise les notions de boucle et de test, qu'il est court et intéressant[2].

[1] mais il l'est pour démontrer la convergence, en utilisant le théorème des accroissements finis, lequel est d'ailleurs hors programme...

[2] par contre il est basé sur une confusion entre suite convergente et suite de Cauchy...

Langages compilés

Les langages C et C++

Le langage C est incontestablement le plus répandu. Surtout si on l'assimile à son extension "objets" appelée C++. Parmi les très nombreux logiciels écrits en C ou C++, on peut citer tous les tableurs connus, les navigateurs Internet, les systèmes d'exploitation, Maple, xcas, etc. etc. etc. Bref, la plupart des utilisateurs d'ordinateur se servent exclusivement de logiciels écrits en C ou C++...
L'exemple présent est suffisamment simple pour qu'on n'ait pas besoin d'utiliser la fonction "malloc" qui oblige à calculer la taille de mémoire dont le programme aura besoin, tâche qui ne simplifie pas la programmation... Les boucles simples, inutilisées ici, s'écrivent "for(n:=0;n<=10;n++)" pour faire aller "n" de 0 à 10, ce qui n'est pas extrêmement simple. Mais on s'y fait, et à ce moment là on devient ***productif***, dans le sens où

- on prend une valeur sur le marché de l'emploi (C est le langage de l'industrie)
- on est aussi adapté à d'autres langages comme Java, Javascript, etc. et même certains logiciels de calcul formel tels *xcas* et *yacas*...

Seulement il faut pour cela "franchir le pas", ce qui amène à la question "qui doit former les lycéens à la programmation, profs de maths ou autres?" à laquelle la réponse est délicate: D'un côté il y a l'impératif du temps, et de ce point de vue on va voir plus bas que C n'est pas l'idéal, d'un autre la peur d'avoir l'air ridicule en ayant investi du temps dans un outil qui ne prépare pas du tout les élèves à leur futur emploi... Le problème est d'autant plus complexe que, chaque année, il y a plusieurs élèves de seconde qui savent ***déjà*** programmer en C/C++...
Quand à l'égalité, elle s'écrit "n==0" si par exemple on veut tester si "n" est

nul.
L'exemple "babylone" aboutit ici au fichier "babylone.c" qui contient le texte suivant:

```
#include <stdio.h>
#include <math.h>

void main()
{
    float r=1,t;
    float s=2/r;
    while (fabs(r-s)>=1e-8)
    {
        t=(r+s)/2;
        r=t;
        s=2/r;
    }
    printf("%1.8f \n" , r);
}
```

Outre le fait qu'il a fallu inclure des fichiers ("stdio" pour pouvoir imprimer le résultat, et "math" pour pouvoir calculer une valeur absolue) et détailler le type des variables utilisées (ici "float" c'est-à-dire réel avec peu de précision), on n'a rien tant qu'on n'a pas ***compilé*** le fichier, par exemple en tapant dans une console[3] le texte "gcc babylon.c" lequel produit un fichier exécutable appelé "a.out", qu'on peut exécuter... (les exécutables sont les fichiers possédant l'extension "sit" sous Mac, "exe" sous Windows et "bin" sous Linux; "gcc" est le compilateur Linux) Et c'en exécutant ce logiciel dans une console qu'on voit apparaître la réponse 1.41421354 qu'on désirait.

[3] inutile avec un environnement de développement tel que Netbeans, Geany, Devc++ etc. pour lesquels il suffit de cliquer sur "compile"

Le langage java

Mis au point par une équipe de Sun qui carburait au café lors de longues et productives nuits blanches, ***Java***[4] a été conçu dès l'origine pour être à la fois un langage objet et un langage orienté Internet. Ce qui explique sans doute pourquoi les logiciels écrits en Java sont souvent des logiciels faisant appel à l'interactivité et au graphisme. Parmi ceux-ci, on peut citer notamment *Freemind*, *Xlogo*, *CaRMetal*, *GeoGebra*, *JasTex* etc. Et c'est Java qui a été choisi par la NASA pour leurs logiciels de présentation. Le principal avantage qu'il y a à programmer en Java est la *multiplateformité*: Le même logiciel en Java peut tourner indifféremment sur Mac Windows Linux... et même sur Nokia !
Pour autant le langage est un langage objet et même avec des environnements tels que *BlueJay*, *NetBeans* ou *OpenJDK* la programmation n'est pas aisée: Les boucles s'écrivent "for(int n=0;n<=10;n++)" et l'égalité ""n==0".

```
package babylone;
public class Main {

    public static void main(String[] args) {
        float r,s,t;
        r=1;
        s=2/r;
        while(Math.abs(r-s)>=1e-8){
            t=(r+s)/2;
            r=t;
            s=2/r;
        }
        System.out.println(r);
    }

}
```

La première ligne est superflue, elle a été créée par l'environnement *NetBeans* qui raisonne en terme de "projets", c'est-à-dire des dossiers où sont réunies les nombreuses sources du programme, dans le cas présent inutiles car il n'y a qu'une source...

[4] nom issu d'une erreur phonétique sur "kawa" que l'un des développeurs, francophile, avait retenu pour "café"; l'icône représentant Java est toujours une tasse de café...

Langages interprétés

Basic

Le nom de ***BASIC*** est un acronyme signifiant ***Beginner's All purpose Symbolic Instruction Code***, bref, langage pour débutants. La version choisie ici s'appelle *yabasic* et est à la fois légère et multiplateforme (Windows et Linux). L'égalité s'écrit "n=0" et les boucles "for n=0 to 10". Le script s'écrit ainsi:

```
r=1
s=2
repeat
   r=(r+s)/2
   s=2/r
until(abs(r-s)<0.00000001)
print r
```

Le script produit la sortie "1.41421".

Perl

Perl est-il compilé ou interprété? Il semble que ce soit une sorte de C++ interprété, en tout cas ça y ressemble. La mise au point d'un programme en *Perl* est assez rapide, mais les variables doivent être précédées du symbole "$" pour ne pas être confondues avec leur nom. En *Perl*, l'égalité se note "$n==0" et les boucles "for($n:=0;$n<=10;$n++)". Le calcul de la racine de deux devient ici:

```
#!/usr/bin/perl
$r=1;
$s=2/$r;
while (abs($r-$s)>1e-8)
{$r=($r+$s)/2;
$s=2/$r;
}
print $r;
```

Il a fallu expliquer à Perl où se trouve Perl (première ligne)! Et en console, l'exécution (après compilation?) affiche directement 1.41421356237469.

Python

Langage très populaire mais quelque peu réservé au monde de Linux, Python est assez agréable à utiliser grâce à son interpréteur. Toutefois la structure du programme est représentée par l'indentation, ce qui allège les scripts mais les rend quelque peu difficiles à lire...
En Python les boucles s'écrivent "for n in range(1,10):" et l'égalité "n==0".

```
r=1.0
s=2.0/r
while abs(r-s) >= 1e-8:
    t=(r+s)/2
    r=t
    s=2/r
print r
```

Python ne nécessite pas qu'on précise si r est entier ou réel, mais si on écrit "r=1", r est automatiquement considéré comme entier, et Python plante à la ligne "r=t" où on essaye de mettre une fraction dans un entier: ça ne passe pas... D'où le "r=1.0" qui fait de r un réel.

php

Le langage à la mode !... En fait php est incorporé dans un autre langage, le *html*, est basé sur un autre langage, le *javascript* (voir plus bas) et utilise les bases de données avec *MySQL*. En plus il faut un site Internet pour pouvoir faire tourner un script php dessus, et pour couronner le tout, il est difficile de prévoir l'avenir qu'aura ce langage (le langage ".net" qui devait être l'unique langage de l'avenir, ne semble pas avoir eu le succès escompté...). En php, les boucles s'écrivent "for $n=0; $n<=10; $n++" et les tests "n==0".

```
<html><head>
</head>
<body>
<?php
$r=1.0;
$s=2.0/r;
while(abs($r-$s)>=1e-8)
    {
        $t=($r+$s)/2;
        $r=$t;
        $s=2/$r;
    echo $r . "<br/>";
    }
?>
</body>
</html>
```

Comme Python, php attribue automatiquement un type à chaque variable une fois que celle-ci prend une valeur; donc comme avec Python, il faut écrire "r=1.0" et non "r=1" qui eût fait de r un entier.

Javascript

sur site internet

Le langage ***Javascript***, à ne pas confondre avec java, est un langage dont les trois principaux interpréteurs s'appellent *Internet Explorer*, *Mozilla Firefox*

et *Netscape Opera*. Autant dire que tout le monde en a sur son ordinateur même si on ne le sait pas toujours. Le programme Javascript doit s'écrire à l'intérieur d'un fichier "htm" ou "html" entre des "balises" $<$ *script language* $=$ *Javascript* $>$ et $<$ */script* $>$. Javascript est très fort en fenêtres, boîtes de dialogues, boutons à cocher mais apparemment pas en graphisme. Les boucles s'écrivent "for (n=0;n<10;n++)" et l'égalité "n==0".

```
<html><head>
</head>
<body>
<script language="Javascript">
    r=1;
    s=2/r;
    while (Math.abs(r-s)>1E-8){
        t=(r+s)/2;
        r=t;
        s=2/r;
    }
    document.write(r);
</script>
</body>
</html>
```

avec CaRMetal

Bien que ***CaRMetal*** soit un logiciel de géométrie dynamique, à partir de la version 3 il est muni d'un éditeur de texte javascript, avec le confort de l'indentation automatique, du choix des instructions dans une palette et de la correction syntaxique. En plus il possède des instructions graphiques qui rendent plus aisées les représentations graphiques. Ainsi c'est le seul des exemples de ce comparatif qui a été mis au point en moins de 20 secondes...

```
r=1;
s=2/r;
while (Math.abs(r-s)>1E-8){
    t=(r+s)/2;
    r=t;
    s=2/r;
}
Print(r);
```

Ruby

Ruby possède son interpréteur en ligne.
En Ruby les boucles s'écrivent "10. times do ... end" et l'égalité "n==0"

```
r=1.0
s=2.0/r
while (r-s).abs>=1e-8
t=(r+s)/2
r=t
s=2/r
end

r
```

On remarque que les fonctions sont écrites après les variables comme dans "(r-s).abs" pour avoir $|r-s|$

Lua

Lua (lune en portugais, en effet ce langage est brésilien) est un langage de script très peu connu et très peu utilisé. Mais il a été choisi pour écrire des variantes de certains jeux de stratégie, et notamment le jeu ***Enigma***, l'un des premiers

jeux libres (sous license GPL) est entièrement écrit en *lua*. À propos de ce jeu, quelques-uns de ses "niveaux" font directement appel à l'algorithmique:

- "The Turtle" (III-45)
- "Le jeu de la vie" (IV-28)
- "MetaPuzzle" (IV-95)
- "Print23" (V-23)

En plus, les labyrinthes sont tous construits automatiquement avec l'algorithme de Kruskal qui est un algorithme de parcours d'arbres...
En *lua*, les boucles s'écrivent "for n = 0 , 10 do ... end" et l'égalité "n==0". Le langage est si peu répandu qu'un interpréteur en ligne permet de le tester sans l'installer: On peut l'essayer ici.

```
r=1
s=(r+2/r)/2
while (r-s)^2>1e-16 do
    r,s = s,(s+2/s)/2
end
io.write(r)
```

Les logiciels de calcul matriciel

L'un des plus célèbres des logiciels de calcul numérique, ***MatLab***[5], est un logiciel spécialisé dans le calcul matriciel. C'est dû au fait que la très grande majorité des calculs sont linéaires. ***MatLab*** n'étant pas libre, ne sera pas traité ici. Cependant il possède des "clones" libres qui héritent de sa puissance:

[5] d'ailleurs le "Mat" veut dire "matrice" et non "mathématiques"

SciLab

Le logiciel ***SciLab*** est presque totalement compatible avec *MatLab*; il est français et libre, et il y a une version lycée... Scilab est fourni avec un bon éditeur de textes appelé *SciPad*. Les boucles de base s'écrivent "for n=1:10" et l'égalité "n==0". Scilab est plutôt bon pour représenter graphiquement des suites et des fonctions, mais il faut "entrer dans l'optique des vecteurs" pour cela.

```
r=1;
s=2/r;
while abs(r-s)>1e-8;
  t=(r+s)/2;
  r=t;
  s=2/r;
end;
r
```

Euler

Le logiciel ***Euler Math Toolbox*** offre de très nombreux avantages: langage de programmation à la fois simple et puissant, facilité à représenter des fonctions (par exemple, "plot2d("x^2",xmin=-2,xmax=2)"), qualité des graphiques (je l'utilise pour préparer mes cours), de nombreux exemples "orientés lycée" et la présence de deux moteurs de calcul formel. ***Euler*** est libre et une ancienne version tourne encore sous Linux. Mais la version la plus récente n'existe (pour l'instant) que sous Windows. Les boucles se notent "for n=1 to 10" et l'égalité "n=0". Euler est très bon pour représenter des suites et des fonctions mais pour les suites il faut manipuler des vecteurs.

```
>function rac2(x=1)
  $r:=x;
  $s:=2/r;
  $repeat;
    $if abs(r-s)<1e-8; then break; endif;
    $t:=(r+s)/2;
```

```
    $r:=t;
    $s:=2/r;
  $end;
  $return r;
$endfunction
>rac2(1)
```

R

R est un logiciel libre de calcul matriciel, qui s'est vu avec le temps garnir de fichiers de statistiques (données et fonctions) qui en ont fait un logiciel de statistiques. Mais il possède un langage de programmation. Les boucles s'écrivent "for n in 1:10" et l'égalité "n==0".
Par contre *R* ne semble pas posséder de boucle "while" et il n'est visiblement pas aisé d'ouvrir un fichier R dans R...

Octave

Octave est aussi très proche de matlab. Mais il est libre. Le graphisme est intéressant. La programmation plutôt facile. Les boucles s'écrivent "for n=1:10" et l'égalité "n==0".

```
r=1;
s=2/r;
while abs(r-s)>1e-8
  t=(r+s)/2;
  r=t;
  s=2/r;
end
r
```

Les langages de l'IA

Prolog

Prolog est un excellent langage de programmation, qui ne laisse personne indifférent: Il y a ceux qui adorent (rares) et ceux qui ne supportent pas... Création française, le langage est particulièrement adapté à la gestion de bases de données, et à la résolution automatique de problèmes. En Prolog, les affectations s'écrivent "N is 0". Le code peut se traduire par:

$$Suivant(x,y) \Leftarrow y = \frac{x + \frac{2}{x}}{2}$$

$$x \simeq \sqrt{2} \Leftarrow Suivant(x,y) \wedge |x - y| < 10^{-8}$$

$$x \simeq \sqrt{2} \Leftarrow Suivant(x,y) \wedge y \simeq \sqrt{2}$$

ce qui donne le fichier Prolog suivant, où on a ajouté les "effets de bord" constitués de l'écriture des résultats partiels et du "cut" qui empêche une boucle sans fin:

```
suivant(X,Y) :-
    Y is (X+2/X)/2.
rac2(X) :-
    suivant(X,Y),
    abs(X-Y)<1e-8,
    writeln(X).
rac2(X) :-
    suivant(X,Y),
    rac2(Y),!,
    writeln(X).
```

L'exemple de Prolog a sa place dans les langages de l'IA mais c'est aussi un langage compilé avec *Strawberry Prolog*, *swi prolog* (l'exemple présent) et *gprolog*, libre et de l'INRIA.

LISP

LISP est un langage complètement atypique, qui ne connaît qu'une seule sorte d'objets: la liste. Né à la fin des années 50 au MIT, il est encore aujourd'hui très largement utilisé: Tous ceux qui prétendent que LISP est dépassé feraient bien de voir en quel langage on a programmé *emacs*, *Derive*, *Maxima*, *yacas*, *TexMacs* et quel langage de script est utilisé pour *audacity* et le *Gimp*... (en effet, *Scheme* est une variante de *LISP*) Par exemple, sous le Gimp, on peut ouvrir une console LISP en cliquant sur "Extensions>Script Fu"...
LISP est lui aussi dépourvu de boucles itératives autres que "do" qui présuppose le nombre d'itérations, et l'exemple habituel sera donc traité avec la récursivité, en l'occurrence avec la console GIMP:

```
(define rac2
   (lambda (r)
   (if (< (abs (- r (/ 2 r))) 1e-8)
        r
        (rac2 (/ (+ r (/ 2 r)) 2)))))
(rac2 1)
```

LOGO

Apparu lui aussi au MIT au début des années 60, ***LOGO*** ressemble beaucoup à LISP avec sa récursivité. Mais parmi les légères différences entre ces deux langages, on constate

- l'importance du graphisme dans LOGO
- la francisation de LOGO (avec Xlogo par exemple)
- le rapprochement avec le langage naturel
- le côté "langage pour débutant" intéressant en pédagogie; en effet LOGO est le seul langage de programmation à avoir été testé avec succès en école maternelle...

Les boucles s'écrivent "repete 10" et l'égalité "n=0".
Pour l'exemple avec *Xlogo* on commence par écrire "fixedecimales 8" dans l'interpréteur puis

```
pour rac2 :r
tantque [non egal? :r suivant :r] [donne "r (somme :r 2/:r)/2]
ecris :r
fin
```

et enfin, "ecris rac2 1" pour avoir le résultat. À noter que "fixedecimales 30" avant l'opération donne un résultat encore plus précis ! Plutôt que la notation "polonaise" (somme :s 2/:s)/2 on aurait pu mettre "(s+2/s)/s" mais le mot "somme" est plus "grammatical". On remarque que ce programme utilise la récursivité plutôt que les boucles... On remarque aussi que *Xlogo* est très orienté "collège" mais on a d'autres LOGO sur le libre: kTurtle sur kde, et la tortue de *xcas* (oui, même ça il peut faire !). L'éditeur de texte de *Xlogo* n'indente pas automatiquement (il n'y a pas vraiment besoin avec ce langage) mais utilise une coloration syntaxique du plus bel effet.

Calcul formel

Les logiciels de calcul formel, comme Mathematica et Maple, ont souvent aussi un langage de programmation.

Yacas

Aujourd'hui la principale caractéristique de ***yacas*** est qu'il est incorporé dans GeoGebra, permettant à celui-ci de faire un peu de calcul formel. Les boucles s'écrivent comme en C: "for(n=:1;n<=10;n++)" et l'égalité "n==0".

```
r:=1;
s:=2/r;
Until (Abs(r-s) < 10^(-8)) [t:=(r+s)/2; r:=t; s:=2/r; ];
r
```

On a la réponse exacte en fraction: 665857/470832

GP

Graphical Pari est le nom d'une interface pour *Pari*, logiciel de calcul formel peu puissant mais le langage de programmation est très réputé. Par exemple, c'est le langage utilisé pour faire tourner le site *Wims*. L'interpréteur *gp* est extrêmement léger et utilise une coloration syntaxique dans une console. Un vrai bijou. En plus il est français... Les boucles s'écrivent "for(n,1,10,...)" et l'égalité "n==0".

```
r=1
s=2/r
until(abs(r-s)<1e-8,t=(r+s)/2;r=t;s=2/r)
r

 665857/470832
```

On note que, *pari* étant un logiciel de calcul formel, le résultat est donné sous forme exacte (fraction). En effet, $u_n \in \mathbb{Q} \Rightarrow u_{n+1} = \frac{u_n + \frac{2}{u_n}}{2} \in \mathbb{Q}$. On remarque aussi la simplicité du langage. Ce qu'on ne voit pas, c'est la beauté de l'interface.

Maxima

WxMaxima est un logiciel de calcul formel libre et puissant. Il est par exemple incorporé dans *Euler Math Toolbox*, ce qui signifie qu'on peut aussi utiliser la présente méthode dans *Euler Math Toolbox*. L'affectation se note par un simple double-point, les boucles par "for n: 1 thru 10" et l'égalité par "n=0".

```
r:1;
thru 7 do r:(r+2/r)/2;
float(r);
```

On remarque que si on affiche r et non $float(r)$, on a une fraction exacte, assez impressionnante d'ailleurs:

$$\frac{4946041176255201878775086487573351061418968498177}{34973792557579411720208518520705629194379642126 08}$$

On remarque également que *Maxima* ne connaît pas de boucles évoluées avec *while* ou *until*.

xcas

xcas est une interface pour la bibliothèque de calcul formel *giac*, mais c'est surtout un logiciel généraliste: Tortue LOGO, modes *pari*, *Maple*, *ti89* et même un peu de géométrie dynamique, y compris dans l'espace. Les affectations sont notées "n:=0", les boucles "for(n=0;n<10;n++)" comme dans le langage C. Les programmes peuvent être entrés dans l'éditeur de xcas:

```
r:=1;
s:=2/r;
while(abs(r-s)>1e-8)
{
  t:=(r+s)/2;
  r:=t;
  s:=2/r;
}
```

xcas donne lui aussi un résultat exact sous forme de fraction: $\frac{941664}{665857}$

Langages à dominante graphique

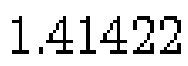
LOGO, déjà vu ci-dessus, est un langage essentiellement graphique. De même, les langages *Scheme* (Gimp et DrGeo), *Python* (kig) et *javascript* (CaRMetal) sont utilisables dans un contexte franchement graphique. Dans cette section on va voir des langages qui sont spécialisés dans le graphisme, et donc à la fois ludiques et utiles notamment en géométrie.

MétaPost

Ce langage un peu ancien a servi à construire des figures vectorielles (exemple: le symbole "€" qui est une figure géométrique). Son principal inconvénient est le manque de précision: 16 chiffres binaires et c'est tout. Il est donc plus adapté aux entiers qu'aux réels. Mais pour des courbes de Bézier et fractales diverses, ce langage est le champion !
En MetaPost, les tests s'écrivent "n=0" et les boucles "for n=0 upto 10: ... endfor". Le script est le suivant (on a remplacé la condition de sortie de boucle par "r=s" puisque le langage ne connaît pas le calcul à grande précision):

```
r=1;
s=2/r;
beginfig(1);
forever:
r:=(r+s)/2;
s:=2/r;
exitif (r=s);
endfor
label.rt(decimal r, (0,0));
endfig;
end.
```

En fait *MetaPost* est considéré comme un langage compilé, et la "compilation" produit un fichier "postscript" que voici:



tcl/tk

Le langage ***tcl/tk*** est spécialisé dans les diagrammes et la gestion de la souris. Une interface existe d'ailleurs en *SciLab* pour ce langage de script.
En *tcl/tk* l'égalité s'écrit "n==0" et les boucles "

```
{for {set n 1} {$n<=10} {incr n}}
```

```
%set r 1.0
   1.0
%set s [expr 2/$r]
   2.0
%while {[expr abs($r-$s)]>0.00000001} {
set r [expr ($r+$s)/2]
set s [expr 2/$r]
}
%puts $r
   1.41421356238
```

Asymptote

Amélioration du langage ***MetaPost***, Asymptote est libre et puissant. Sur le site internet on voit des exemples dont certains animés, qui impressionnent vraiment !
Dans Asymptote, l'égalité s'écrit "n==0" et les boucles
"for(int n=0; n < 10; ++n)".
Le script Asymptote ressemble à ceci:

```
real r,s
r=1
s=2/r
while(abs(r-s)>1e-8){r=(r+s)/2;s=2/r;}
s
```

et affiche "1.4142135623715".

POV

Incontestablement le meilleur éditeur de texte (du moins dans la version Windows), POV ("Persistence of vision") est un moteur de rendu, utilisé par exemple pour le film "Dimensions". Mais puisqu'il a un langage de script, autant

l'utiliser (en entrant l'option "Debug_Console=On"). Les tests s'écrivent "n=0" et la seule structure de boucle est la boucle "while" avec laquelle on peut simuler un "for...to". On écrit le script suivant:

```
#declare r=1;
#declare s=2/r;
#while(abs(r-s)>1e-8)
    #declare r=(r+s)/2;
    #declare s=2/r;
#end
#debug str(s,0,8)
```

On clique sur "rendu" et on voit apparaître une image noire; pour une fois on ne s'intéresse pas à cette image, mais on lit dans la console le nombre 1,41421356. Variante, avec le rendu:

```
#include "colors.inc"

#declare r=1;
#declare s=2/r;
#while(abs(r-s)>1e-8)
    #declare r=(r+s)/2;
    #declare s=2/r;
#end

camera {
  location  <0.0, 0, -8.0>
  direction 1.5*z
  right     x*4/3
  look_at   <0.0, 0.0,  0.0>
}

light_source {
  <-.1, .2, -10>
  color White
  translate <-30, 30, -30>
}
```

```
difference {
box { <-4, -4, 0.1>, <4, 4, 10> }
text {ttf "arial.ttf",str(s,0,8),.2,0 translate -2.5*x}
  pigment {marble
           color_map {
            [0.1 color rgb <1,0.4,0.4>]
            [0.5 color rgb 1]
                    }
           turbulence 2
           scale 2
          }
  finish {ambient 0.02 diffuse 0.6 phong 8 phong_size 20}
}
```

C'est plus long, mais maintenant le nombre est gravé dans le marbre (des Babyloniens, on est passé aux Hellènes):

L'intérêt essentiel de POV est qu'il permet aussi de représenter aisément des volumes, ce qui est particulièrement intéressant pour les surfaces cartésiennes et pour illustrer les opérations booléennes (intersection, réunion).

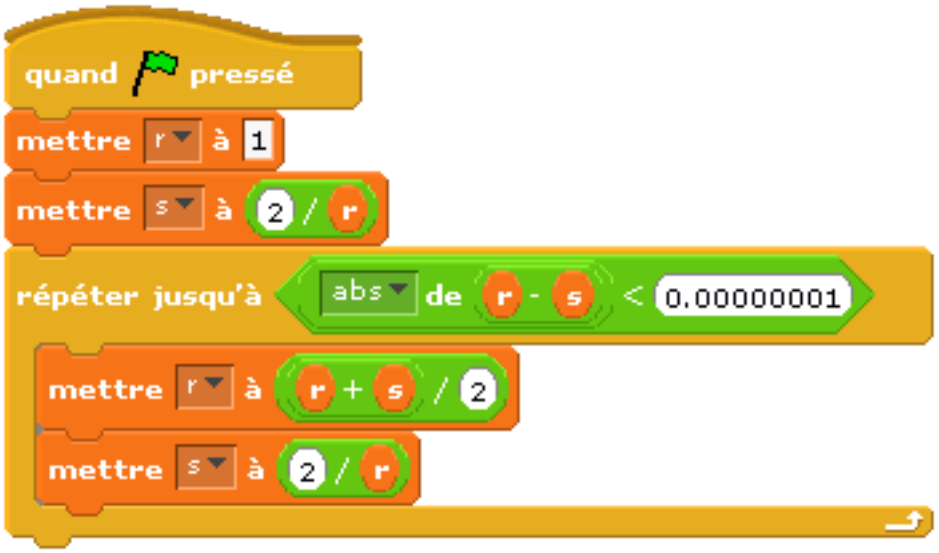
RobotProg

RobotProg est tellement atypique qu'il ne peut pas être utilisé pour calculer la racine de 2, puisque seuls les nombres entiers font partie de ce langage. Toutefois il possède toutes les boucles usuelles, l'égalité s'écrit "n=0" et les boucles "pour n=1 jusque 10": Et oui, ce langage est en français ! En plus l'environnement est géométrique puisqu'il s'agit de déplacer un robot sur un terrain bidimensionnel discret (à coordonnées entières). Et les programmes ne sont pas écrits mais

dessinés, sous forme d'organigrammes. Outil très intéressant donc pour qui veut faire de l'arithmétique et du graphisme (suite de Syracuse, pgcd, nombres amiables etc.)

Scratch

Parce que ***scratch*** a été conçu pour des enfants, la conception d'un programme est la plus agréable de tous les langages de ce comparatif: On déplace et assemble des briques pour constituer un "listing". Ce qui fait de *scratch* un outil immensément valable pour des débutants. Par contre l'affichage est fait en basse précision ce qui prédestine plus ce langage à de l'arithmétique (en fait, à du multimédia). Dans la même catégorie, on trouve les logiciels de traitement de signal: SyncModular (pas libre), Pure Data (multiplateforme car écrit en tk) et SciCos, fourni avec SciLab (encore lui !). En *scratch*, l'égalité s'écrit "n=0" et les boucles "Répéter 10 fois". Le calcul de la racine de 2 donne le "listing" suivant:



Après ça il suffit de cliquer sur le drapeau vert et on voit les affichages de r et s devenir égaux à 1,4. Ce qui est juste mais pas très précis, c'est le moins qu'on puisse dire...

Execalgo

Execalgo, cocorico, est français. Pas du tout spécialisé dans le graphisme, au contraire, il a trouvé sa place ici par une vague ressemblance avec RobotProg... Execalgo est non seulement français mais aussi ***en*** français, que ce soit le langage lui-même ou ses affichages (en mode pas-à-pas par exemple). Execalgo est *recommandé* par l'EN. Par contre il n'est pas multiplateforme ce qui est dommage... Mais le logiciel est suffisamment léger et paraît-il libre, pour qu'on puisse espérer un portage bientôt sous Mac et Linux. Dans Execalgo, l'égalité s'écrit "n=0" et les boucles s'écrivent en français: "[Label début] Donner à n la valeur n+1 Aller à [Label début] si n<10". Le programme pour la racine de deux devient

```
Réel r;s
Donner à r la valeur 1
Donner à s la valeur 2
[Début de la boucle]
Donner à r la valeur (r+s)/2
Donner à s la valeur 2/r
Aller à [Début de la boucle] si abs(r-s)>0.00000001
Afficher r
```

et en cliquant sur "Exécution", on obtient l'affichage suivant:

```
1.41421356
 Exécution terminée.
 Instructions exécutées : 15
```

Execalgo pêche par omission de boucles élaborées comme "tant que" ou "jusqu'à". Ce qui le rend peu propice à l'apprentissage de la programmation structurée tel qu'il est préconisé par Donald Knuth et Jacques Arsac (le même reproche peut être fait aux langages de l'AI). Par contre l'affichage systématique du temps d'exécution est sans doute le meilleur moyen de susciter spontanément des questions sur l'algorithmique (science de la rapidité des algorithmes). Si seulement *Execalgo* faisait du graphisme...

Sevrage

Tableur

L'exemple choisi porte sur les suites. Or un tableur est un outil très pratique pour étudier des suites[6]. En bref, après avoir démarré un tableur, on entre "1" dans la cellule A1, et "=2/A1" dans la cellule B1. Après ça on entre "=(A1+A2)/2" en A2 et on copie B1 en B2. Ensuite il suffit de copier le contenu des cellules A2 et B2 vers le bas: On obtient ceci:

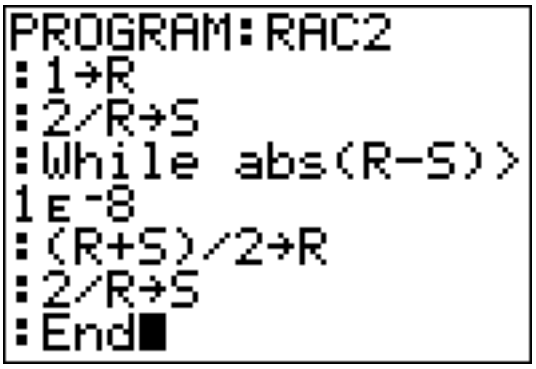
1	2
1,5	1,33333333333333
1,41666666666667	1,41176470588235
1,41421568627451	1,41421143847487
1,41421356237469	1,4142135623715
1,41421356237309	1,4142135623731
1,41421356237309	1,4142135623731
1,41421356237309	1,4142135623731
1,41421356237309	1,4142135623731
1,41421356237309	1,4142135623731

(le tableur *Gnumeric* a été utilisé mais d'autres, comme celui de GeoGebra ou d'xcas feraient aussi l'affaire). On constate que l'usage de tests a été évité mais c'est le prix d'une simplicité non négligeable par rapport à la programmation (les formules ont partiellement été entrées à la souris).

[6]En bref, tout ce qui précède est complètement inutile, mais ça fait parfois du bien de se ridiculiser: ça aide à remettre les idées en place...

Les calculatrices

Les calculatrices graphiques sont aussi programmables, avec un langage proche de *BASIC*. Voici par exemple un programme écrit sur une *ti84plus*:



Il suffit de lancer le programme "rac2" et, quand on voit "Done", d'écrire "R" et on a la réponse (on peut aussi ajouter un "Disp R" à la fin du programme). Comme on le voit la calculatrice est un outil très valable pour ce genre d'activité...