Algorithmique avec Postscript

Le langage Postscript est basé sur une pile (c'est un descendant de Forth qui faisait tout avec une pile) et possède des instructions similaires à celles de la tortue Logo. On peut s'en servir pour rédiger et tester rapidement des algorithmes du style "STMG". On va ici voir comment faire pour résoudre le problème suivant:

Exercice d'algorithmique

Un article vaut 80 € initialement. Quel sera son prix après 10 augmentations successives de 2,5% chacune ?

Pour programmer en Postscript, on a besoin d'un interpréteur; celui-ci s'appelle *ghostscript* et se lance par exemple dans la console d'un système à base d'Unix, en entrant *gs*. Dans cette console, si on entre 80 puis le bouton "entrée", on ne voit rien, sinon un 1 à gauche: Ce 1 veut dire que la pile est actuellement de hauteur 1. En effet on a entré 80 dedans. Une *pile* est une mémoire de nature particulière, où il n'y a pas vraiment d'affectation au sens habituel, mais les données sont stockées dans la pile toujours par le même endroit, qui est le haut de la pile. Après avoir entré 80, la pile ressemble alors à ceci:

Pour terminer correctement un programme Postscript, on doit laisser une pile vide, c'est une question de politesse et de propreté. Pour enlever un élément de la pile, on le tire vers le haut avec *pop*. Alors *80 pop* est un programme Postscript correct puisqu'après avoir posé 80 en haut de la pile, on l'enlève. Mais dans la suite de cet article, au lieu d'utiliser *pop*, on préférera == qui a le même effet, mais en plus il affiche l'élément dans la console.

Si on veut calculer 2,5% de 80 €, on doit évidemment entrer aussi les 2,5 sur la pile. Alors, après avoir entré et exécuté `80 2.5`, la pile ressemble à ceci:

Si on entre ensuite 100, le script `80 2.5 100` donnera alors à la pile une hauteur de 3:

Ensuite, on applique les transformations suivantes à la pile:

1. `div` qui va remplacer les deux derniers nombres de la pile par leur quotient:

2. `mul` qui va remplacer les deux nombres du haut de la pile par leur produit:

Le script `80 2.5 100 div mul ==` affiche donc 2 et 2,5% de 80 € font effectivement 2 €.

Pour gagner encore en concision pour la suite, on va créer une *fonction* de Postscript. Cela consiste juste à dire que la suite `100 div mul` s'appellera `pourcents`. On définit alors le mot `pourcents` comme abréviation de cette suite d'instructions, avec `/pourcents { 100 div mul } def`. Le calcul du pourcentage se fait alors avec `80 2.5 pourcents ==`.

Pour additionner l'augmentation aux 80 €, on a un petit problème: Les 80 ont été absorbés par le calcul et ne sont donc plus dans la pile. On a donc besoin, dès le début, de dupliquer les 80[1]. `80 dup` fait donc la même chose que `80 80`. Ce qui fait que `80 dup 2.5` donne cette pile:

[1] C'est ce qu'on appelle *clonage* dans le programme du collège

Alors `80 dup 2.5 pourcents` donne cette pile:

Il reste alors à ajouter `add ==` ce qui va successivement remplacer la pile par la somme de ses deux éléments puis vider la pile en affichant la somme. Voilà donc comment on peut augmenter 80 de 2 % avec Postscript:

Augmentation de 2,5 %

```
/pourcents { 100 div mul } def
80 dup 2.5 pourcents add ==
```

Pour continuer à se simplifier la vie, on peut définir comme augmentation annuelle tout ce qu'on a fait subir à 80 (tout ce qui est entre 80 et ==). On va définir ça comme une augmentation et il suffira alors de faire `80 augmenter ==` pour avoir les 82:

Une augmentation (de 2,5 %)

```
/pourcents { 100 div mul } def
/augmenter { dup 2.5 pourcents add } def
80 augmenter ==
```

Pour deux augmentations successives on peut faire `80 augmenter augmenter ==` mais pour 10 augmentations successives c'est un peu long. Alors on va demander à Postscript de répéter 10 fois l'augmentation:

10 augmentations

```
/pourcents { 100 div mul } def
/augmenter { dup 2.5 pourcents add } def
80 10 { augmenter } repeat ==
```