Xcas et son module de programmation

Xcas [1] est un logiciel libre de calcul formel.
Les exemples ci-dessous présentent la syntaxe des principales instructions de programmation de Xcas. L'entrée dans l'éditeur de programme se fait avec la commande Alt+P.

1 La suite de Fibonacci

Léonard de Pise, fils de Bonacci, s'est intéressé en 1202 au fameux problème des lapins : un couple donne naissance, à partir du deuxième mois à un couple chaque mois, les nouveaux couples suivant la même loi de reproduction ; combien y aura-t-il de lapins (supposés immortels...) au bout de n mois ?
Il s'agit donc d'étudier la suite (F_n) définie par :

$$F_0 = F_1 = 1 \qquad F_{n+2} = F_n + F_{n+1}$$

On peut écrire la (double !) récursion directement :

```
fiborec(n):={ //déclaration de la fonction
if n<2
   then return(1);
   else fiborec(n-1)+fiborec(n-2);
end_if
}:;
```

Listing 1 – suite de Fibonnacci en récursif brut

Très facile....mais peu efficace car il faut une vingtaine de secondes à XCAS pour calculer `fibo(30)`. En effet, la récursion est double et nécessite une pile très importante. Mieux vaut recourir à une fonction impérative [2].
On obtient les 15 premiers termes en faisant *seq(fiborec(j),j=0..14)*.

```
fiboimp(n):={
local q;
local fib:=[1,1];// création d'une liste dont les deux premiers termes
     sont 1 et 1.
for q from 2 to n-1 by 1 do
    fib[q]:=fib[q-1]+fib[q-2];
    end_for;
return(fib);
}
:;
```

Listing 2 – suite de Fibonnacci en impératif avec XCAS

La commande *fiboimp(15)* donnera la liste des 15 premieres valeurs.

1. Téléchargement sur http://www-fourier.ujf-grenoble.fr/~parisse/irem.html
2. ou une fonction récursive terminale via une fonction auxiliaire.

2 Exercices

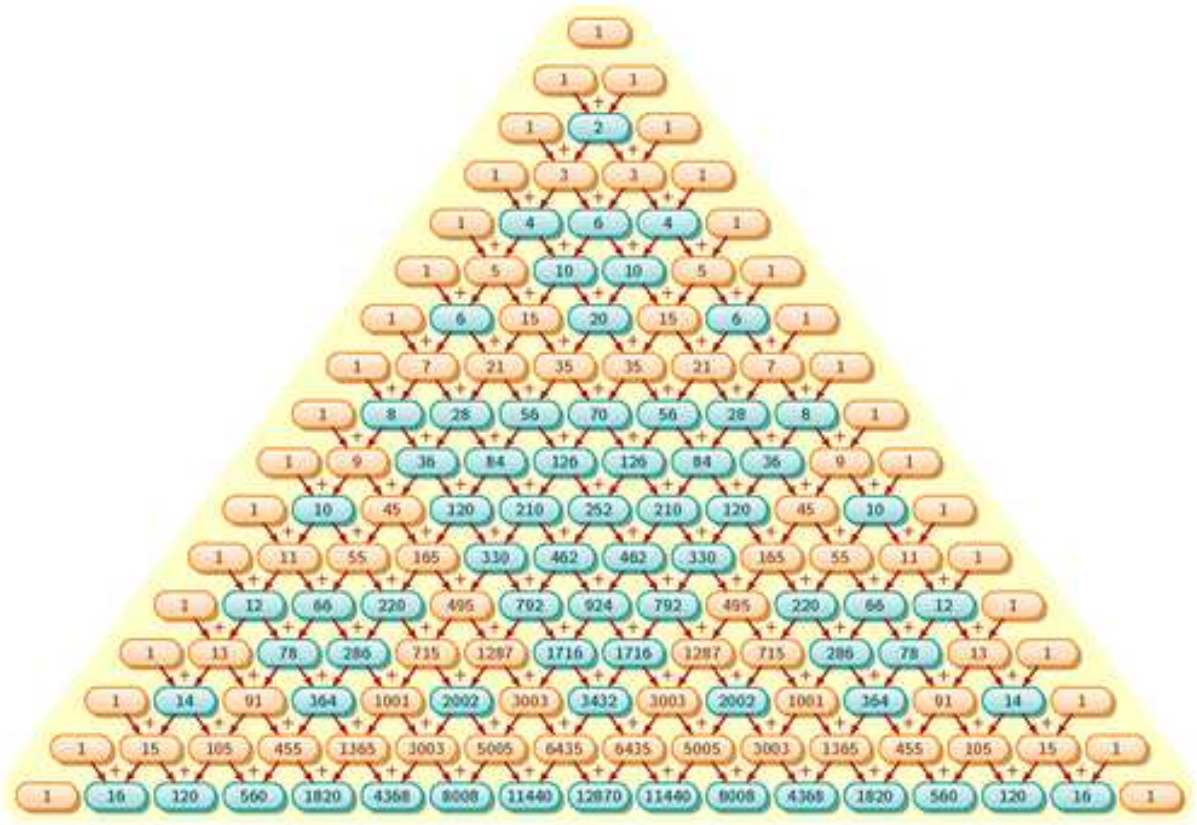
1 On peut définir $n!$ en disant que $1! = 1$ et que $n! = n.(n-1)!$

1. Ecrire une fonction récursive et une fonction itérative pour calculer $n!$. Comparer les deux fonctions en termes d'efficacité.
2. Utiliser la fonction précédente pour définir les fonctions A_n^p et C_n^p.

2 Écrire une fonction retournant le triangle de Pascal jusqu'a un entier n. On utilisera une matrice $n \times n$ pour représenter un tableau et on utilisera la formule de pascal $C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$ pour calculer les termes différents de 1.

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
```

3 Modifier le programme précédent afin de remplacer tous les nombres pairs par une étoile et tous les nombres impairs par un blanc.



Xcas et son module de programmation

```
factorielle(n):={
local fact:=1;// création d'une variable locale (non typée).
for q from 1 to n by 1 do
    fact:=fact*q;
    end_for;
return(fact);
}
:;
```

Listing 3 – Factorielle en impératif avec XCAS

```
factoriellerec(n):={
if n<2
   then return(1);
   else return(n*factoriellerec(n-1));
end_if
}
:;
```

Listing 4 – Factorielle en récursif avec XCAS

```
arrangement(n,p):=
{return(factorielle(n)/factorielle(n-p));
}
:;

combinaison(n,p):=
{return(factorielle(n)/(factorielle(n-p)*factorielle(p)));
}
:;
```

Listing 5 – Arrangement et combinaison

```
Pascal(n):={
local T,j,i;
T:=matrix(n,n);
pour j de 0 jusque n-1 faire
     pour i de 0 jusque n-1 faire
     si j>i alors T[i,j]:="" ;
            sinon si j==0 alors T[i,j]:=1 ;
            sinon si i==j alors T[i,j]:=1;
            sinon si (i>0) et (j>0) alors
                  T[i,j]:=T[i-1,j]+T[i-1,j-1];
            fsi;fsi;fsi;fsi;
    fpour;
fpour;
retourne(T);
}
:;
```

Listing 6 – Triangle de Pascal

Pour le triangle fractale, il suffit d'ajouter juste avant l'instruction *return* finale :

```
pour i de 0 jusque n-1 faire
   pour j de 0 jusque i faire
       si irem(T[i,j],2)==0
          alors T[i,j]:="";
          sinon T[i,j]:="*";
       fsi;
   fpour;
fpour;
```