# Programmes de base pour le calcul du factoriel

## II.1- Approche récursive :

La récursivité, d'un point de vue théorique reste assez simple , il s'agit de programmes ou de fonctions d'un programme qui ont la faculté de s'appeler eux-mêmes (on entend également le terme *d'auto-appel* ).

La récursivité est une manière simple et élégante de résoudre les problèmes algorithmiques, notamment en mathématique du calcul du factoriel mais cela ne s'improvise pas, il convient donc de savoir comment ce principe fonctionne afin de conclure a propos de sa puissance et capacité pour aboutir à satisfaire notre besoin.

Nous avons programmé alors deux types de fonctions récursives: les fonctions récursives simples  et les fonctions récursives terminales. Et nous avons pu ainsi déduire leur limites dans le calcul des factoriel des grands nombres.

### II.1.1- les fonctions récursives simple :

Nous pouvons définir notre règle de calcul de cette manière :

*n ! = (n) (n-1) (n-2) ... (1) . (1)*

Nous pouvons en déduire que nous avons fait des appels en décrémentant la valeur de *n*  à chaque appel de la fonction jusqu'à ce que *n == 1* !

Ce qui nous donne la fonction suivante :

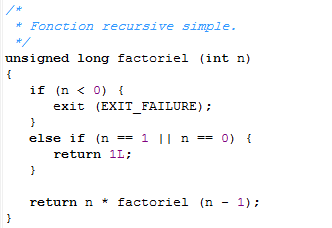


Figure 1 : code de focntion récursive simple

### II.1.2- *les fonctions récursives terminales :*

Une [fonction](http://fr.wikipedia.org/wiki/Fonction_(informatique)) à récursivité terminale (dite tail-recursive en anglais) est une fonction où l'appel récursif est la dernière [instruction](http://fr.wikipedia.org/wiki/Instruction) à être évaluée. Cette instruction est alors nécessairement « pure », c'est-à-dire qu'elle consiste en un simple appel à la fonction, et jamais à un calcul ou une composition.

Contrairement à la récurions normale que sa dernière instruction est une composition faisant intervenir l'appel récursif.

Dans les fonctions récursives terminales, aucune référence aux résultats précédents n'est à conserver en mémoire, tandis que dans la simple récursivité, tous les résultats intermédiaires doivent l'être. Les algorithmes récursifs exprimés à l'aide de fonctions à récurions terminale profitent donc d'une optimisation de la [pile d'exécution](http://fr.wikipedia.org/wiki/Pile_d%27ex%C3%A9cution).

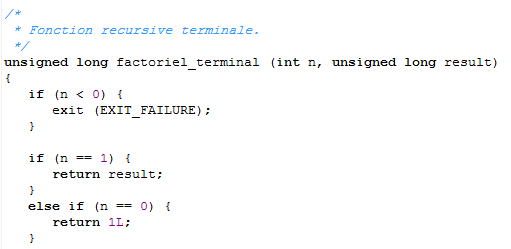


Figure 2 :la fonction récursive terminale

### II.1.3- Remarques et Conclusions:

* Quand on travaille sur de très grands nombres, un phénomène se produit lorsqu’on essaye de stocker un nombre plus grand que ce que peut contenir le type de votre variable c’est le dépassement de capacité.
* Il est préférable ainsi d’évitez le type int en travaillant avec une fonction récursive, pour le calcul de grands factoriel. Ce type est très petit et dans une fonction récursive il peut très vite arriver à le dépasser et c'est donc le plantage assuré du PC. Un type comme long est plutôt préféré pour assurer un minimum la viabilité du code.
* Il faut une condition de sortie pour la fonction, mais il faut être très vigilant quant au choix de la condition, on doit être sûr qu'elle soit validée à un moment ou à un autre sinon on entre dans une boucle infinie sans condition de sortie.
* Les appels récursifs de fonctions sont placés dans la pile du programme, pile qui est d'une taille assez limité car elle est fixée une fois pour toutes lors de la compilation du programme. Dans la pile sont non seulement stockés les valeurs des variables de retour mais aussi les adresses des fonctions entre autres choses, les données sont nombreuses et un débordement de la pile peut très vite arriver ce qui provoque sans conteste une sortie anormale du programme.
* Pour toutes ces raisons la récursivité reste une solution bien limité et qui ne satisfait pas notre besoin.

***Note***

Il faut noter que la taille d'un int peut être différent suivant les implémentations systèmes.

En effet, par exemple en 32 bits avec un compilateur Microsoft (c), la taille d'un int est la même que celle d'un long, il est donc préférable de se renseigner sur la taille des variables suivant le système utilisé.

|  |  |
| --- | --- |
|  |  |

## II.2- Approche itérative :

### II.2.1- la fonction itérative :

Un Rappel itératif consiste à répéter n fois un processus en faisant changer la valeur des variables jusqu’a obtention du résultat.

Le Calcul itératif de factorielle d’un nombre : n! =

Un calcul itératif se programme par une boucle (for ou tant-que ou repeat-until)

### 

Figure 3: Fonction Itérative

### II.2.2- Remarque et conclusion :

* L'avantage dans une fonction itérative, réside dans le fait qu’on ne risque pas de débordement de pile mais très certainement un dépassement de capacité qui dans ce cas fait également terminer le programme en retournant le signal SIGFPE qui est une constante standard qui indique diverses opérations mathématiques incorrectes (dans notre cas un dépassement de capacité ).
* L’approche itérative telle une boucle for est bien moins coûteuse en termes de ressources et est bien plus sûre, sauf dans le cas d'un dépassement de capacité.

# Programme basé sur la formule de Stirling pour le calcul du factoriel

 Il existe une méthode qui donne directement l'expression de n! En fonction de n. C'est la formule de Stirling (découverte tout d'abord par De Moivre) .

La formule de Stirling permet une approximation rapide de la valeur de n ! pour de très grands nombres

(2)

Ou encore en C++ :

(3)

## III.1- Approche mathématique de la formule de Stirling :

|  |
| --- |
|  |

[De Moivre](http://serge.mehl.free.fr/chrono/Moivre.html) avait affirmé,[2] dans la première édition de son traité Doctrine of Chance, que pour n infini, le rapport :

  (4)

tendait vers une limite finie k (la notation n!, factorielle n , désignant le produit des n premiers entiers naturels non nuls : n! = 1 x 2 x ...  x n).

[De Moivre](http://serge.mehl.free.fr/chrono/Moivre.html) ne calcula pas la valeur précise de k. Reprenant les travaux de, Stirling, dans un traité publié en 1730 à Rome, Methodus differentialis seu de summatione et interpolatione seriarum développe en série ln G(x), où G est la célèbre fonction d'[Euler](http://serge.mehl.free.fr/chrono/Euler.html#g), ce qui le conduit à compléter la formule d'approximation de factorielle n pour n gra+nd :

(5)

en précisant la nature de r(n), infiniment petit s'exprimant sous la forme d'une série convergente de somme nulle :

) (6)

Stirling établit ainsi que  (mais [Wallis](http://serge.mehl.free.fr/chrono/Wallis.html) l'avait découvert avant lui). C'est dire que l'on a pour n "grand" :

(7)

On a plus précisément encore :

= , 0<(8)

La programmation de la formule de Stirling nous a permis de relever le tableau de valeurs suivant :

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **n!  "exact" selon Matlab** | **n! selon Stirling** | **erreur** |
| 1 | 1 | 0,922 | 7,79% |
| 2 | 2 | 1,919 | 4,05% |
| 3 | 6 | 5,836 | 2,73% |
| 4 | 24 | 23,506 | 2,06% |
| 5 | 120 | 118,019 | 1,65% |
| 6 | 720 | 710,078 | 1,38% |
| 7 | 5040 | 4980,396 | 1,18% |
| 8 | 40320 | 39902,395 | 1,04% |
| 9 | 362880 | 359536,873 | 0,92% |
| 10 | 3628800 | 3598695,619 | 0,83% |
| 11 | 39916800 | 39615625,051 | 0,75% |
| 12 | 479001600 | 475687486,473 | 0,69% |
| 13 | 6227020800 | 6187239475,193 | 0,64% |
| 14 | 87178291200 | 86661001740,599 | 0,59% |
| 15 | 1307674368000 | 1300430722199,470 | 0,55% |
| 16 | 20922789888000 | 20814114415223,100 | 0,52% |
| 17 | 355687428096000 | 353948328666101,000 | 0,49% |
| 18 | 6402373705728000 | 6372804626194310,000 | 0,46% |
| 19 | 121645100408832000 | 121112786592294000,000 | 0,44% |
| 20 | 2432902008176640000 | 2422786846761140000,000 | 0,42% |
| 21 | 51090942171709440000 | 50888617325509700000,000 | 0,40% |
| 22 | 1124000727777607680000 | 1119751494628240000000,000 | 0,38% |
| 23 | 25852016738884976640000 | 25758525370529300000000,000 | 0,36% |

Tableau 4 : Comparaison entre la valeur de factoriel selon stirling et celle de la valeur exacte

D’où a pu tracer l’erreur sur les valeurs de factoriel de grands nombre selon Stirling :

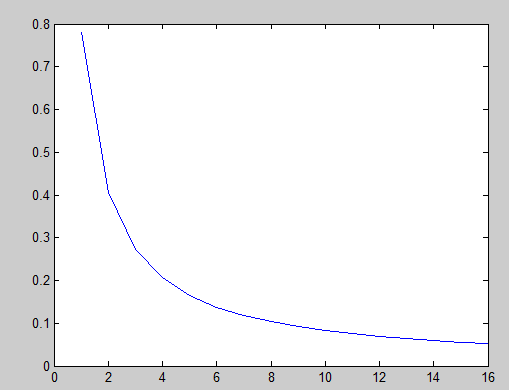


Figure 4:Erreur dégagé entre le resultat du factoriel exact et celui de stirling

* On remarque ainsi que l’erreur sur le calcul de factoriel selon la formule de Stirling tend vers zéro lorsque le factoriel tends vers l’infini.
* Le problème alors reste dans le temps de calcul mis par cette approche qui reste important par rapport a nos exigences.

## III.1-comparaison de l’approche de Stirling et l’approche récursive :

Dans une étape suivante on a commencé à comparer la vitesse d'exécution de l'algorithme récursif et de la formule de Sterling. Pour la gestion du temps en C++.Pour réaliser cette comparaison le code nécessaire:

#include <time.h>       // les deux bibliothèques nécessaires  
#include <sys/timeb.h>

/\* ............................. \*/

Pour comparer le temps d'exécution de deux algorithmes très rapides comme c'est le cas ici, il est nécessaire de faire tourner chacun plusieurs fois. Ainsi, l'appel à l'algorithme sera dans une boucle. On a donc comme résultats :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| factoriel | Resultat stirling | Résultat Récursive | Temps stirling(s) | Temps recursive(s) | Ecart |
| 4 ! | 24 | 24 | 2 | 1 | 0% |
| 5 ! | 119 | 120 | 2 | 2 | 0.83333% |
| 6 ! | 711 | 720 | 2 | 1 | 1.25% |
| 7 ! | 4981 | 5040 | 2 | 3 | 1.17063% |
| 8 ! | 39903 | 40320 | 2 | 2 | 1.03423% |
| 9 ! | 359537 | 362880 | 3 | 3 | 0.921241% |
| 10 ! | 3598696 | 3628800 | 2 | 3 | 0.829586% |
| 11 ! | 39615626 | 39916800 | 2 | 4 | 0.754504% |
| 12 ! | 475687488 | 479001600 | 2 | 5 | 0.691879% |
| 13 ! | 1892272193 | 1932053504 | 2 | 5 | 1.05902% |

Tableau 5: comparaison entre la récursivité et la formule de strirling

Temps moyen de stirling :1.5s

Temps moyen de récursivité : 2.07s

Ecart moyens :0.68%

En réalité la formule de striling que nous avons implémenté a en réalité une différence par rapport au Sterling. Nous avons fait sqrt(2\*pi\*i)\*(i/e)^i + 1.  
Ce (+1) nous a permis d'être exact sur les premiers factoriels (2, 3 et 4) et de rétrécir l'écart de façon générale. En effet, si nous avions strictement utilisé la formule de Sterling, nous aurions par exemple un peu plus de 2% d'erreur pour 5! (119 au lieu de 120), ce qui est bien différent de notre 0,83%.

Cependant, le nombre d'opérations à faire, a été augmentée dans le Sterling. Ainsi, pour un peu plus de concordance, le code est plus lent (il faut rajouter 1, le +1 est une opération supplémentaire qui coûte en terme de temps de calcul).

# Code XXXXX:

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **n!  "exact" selon Matlab** | **n!** | **erreur** |
| 1 | 1 | 1.0000 e+0 | Nulle |
| 2 | 2 | 2.0000 e+0 | Nulle |
| 3 | 6 | 6.0000 e+0 | Nulle |
| 4 | 24 | 2.4000 e+1 | Nulle |
| 5 | 120 | 1.2000 e+2 | Nulle |
| 6 | 720 | 7.2000 e+2 | Nulle |
| 7 | 5040 | 5.0400 e+3 | Nulle |
| 8 | 40320 | 4.0320 e+4 | Nulle |
| 9 | 362880 | 3.6288 e+5 | Nulle |
| 10 | 3628800 | 3.6288 e+6 | Nulle |
| 11 | 39916800 | 3.9917 e+7 | 5.0104e-006 |
| 12 | 479001600 | 4.7900 e+8 | 3.3403e-006 |
| 13 | 6227020800 | 6.2270 e+9 | 3.3403e-006 |
| 14 | 87178291200 | 8.7178 e+10 | 3.3403e-006 |
| 15 | 1307674368000 | 1.3077 e+12 | 1.9601e-005 |
| 16 | 20922789888000 | 2.0923 e+13 | 1.0042e-005 |
| 17 | 355687428096000 | 3.5569 e+14 | 7.2308e-006 |
| 18 | 6402373705728000 | 6.4024 e+15 | 4.1070e-006 |
| 19 | 121645100408832000 | 1.2165 e+17 | 4.0278e-005 |
| 20 | 2432902008176640000 | 2.4329 e+18 | 8.2542e-007 |
| 21 | 51090942171709440000 | 5.1091 e+19 | 1.1319e-006 |
| 22 | 1124000727777607680000 | 1.1240 e+21 | 6.4749e-007 |
| 23 | 25852016738884976640000 | 2.5852 e+22 | 6.4749e-007 |

