

◇ Ecrivez un programme qui calcule $N!$ d'un entier positif quelconque (0 compris).

Solution

Par rapport à l'exercice 2 il y a peu de différences. Pour mémoire on a :

$$n! = \prod_{k=1}^n k = n * (n - 1) * \dots * 2 * 1 \quad \text{avec comme convention } 0! = 1$$

Le programme consistera donc en une boucle, avec comme instruction un produit (cf le corrigé de l'exercice 2). Il faut bien entendu initialiser le produit.

```
#include<stdio.h>

int main(void)
{   int N,fact,k;

    printf("Entrez N : ");
    scanf("%d",&N);

    fact  = 1 ;
    for ( k=2 ; k<N; k++)
    {   fact = fact *k ;
    }

    printf("%d ! = %d \n", N, fact );

    return 0;
}
```

Analyse du programme :

Noter l'astuce consistant à faire débiter la boucle à 2 et non 1, ce qui fait gagner une boucle de calcul. L'initialisation avant la boucle $\text{fact}=1$, assure que l'on ait bien $0!=1!=1$. De plus elle est obligatoire pour avoir un produit exact il faut initialiser à l'élément neutre de la multiplication.

Comme déjà dit précédemment, il **faut procéder à l'initialisation de l'accumulateur juste avant la boucle de calcul**.

◇ Calculez avec votre programme 8! (ou 17! suivant les réglages du compilateur). Que constatez-vous ? Quelle solution proposez-vous pour corriger cette erreur ?

Exemples d'utilisation du programme

Entrez N : 10

10 ! = 362880

Entrez N : 0

0 ! = 1

Entrez N : 17

17 ! = 2004189184

Entrez N : 18

18 ! = -288522240

On constate qu'au delà de 17 le programme retourne un résultat erroné ($n!$ est forcément positif). L'erreur n'est due qu'aux limitations de l'ordinateur.

Un ordinateur code les nombres à l'aide d'une suite de 0 et 1. Pour faire simple le tableau suivant résume les valeurs possibles en supposant que l'on ait des nombres à 4 bits (1 bit = une case

pouvant contenir 0 ou 1)

Représentation Binaire des entiers avec 4 bits

Binaire non signé				
bit 3	bit 2	Bit 1	bit 0	Valeur décimale (base 10)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Binaire signé				
bit 3	bit 2	bit 1	bit 0	Valeur décimale (base 10)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Le bit le plus à gauche (bit3) est le bit dit "de poids fort". En binaire signé, il sert à indiquer le signe.

Cette représentation, simplifiée, comporte quelques difficultés (il y a deux codages pour le nombre zéro en binaire signé).

Deux remarques :

1/ Pour multiplier par deux, il suffit de faire glisser vers la gauche les bits (décalage à gauche), pour diviser par deux, c'est vers la droite que l'on décale.

2/ L'addition s'effectue comme en base 10. Ainsi $0101 + 0010 = 0111$ ($5 + 2 = 7$).

Noter toutefois que $0101 + 0011 = 1000$, soit $5 + 3 = 8$ en binaire non signé, mais on constate qu'en binaire signé on obtient -1 !

Cette erreur est "classique" avec les nombres entiers, qui par défaut sont signés. On peut déclarer les variables comme étant non signées, ainsi :

```
int N,fact,k; devient unsigned int N, fact, k ;
```

et

```
printf("%d != %d \n", N, fact ); devient printf("%d != %u \n", N, fact );
```

Noter le `%u` à la place du `%d` et le `unsigned` devant `int`. Toutefois cela ne résous que très partiellement notre problème. En effet, le résultat n'est plus négatif, mais il est faux ! En ne signant plus les nombres, on multiplie par deux la valeur maximale, or $18! = 18 \times 17!$, on multiplie donc par 18 et non 2, il faudrait rajouter 5bits !!!

La seule solution est de coder la factorielle avec un nombre réel :

```
1 #include<stdio.h>
2
3 int main(void)
4 {   int N,k;
5   float fact;
6
7   printf("Entrez N : ");
8   scanf("%d",&N);
9
10      fact  = 1 ;
11      for ( k=1 ; k<N; k++)
12      {   fact = fact *k ;
13      }
14
15      printf("%d != %E \n", N, fact );
16
17      return 0;
18 }
```

Noter le **%E** comme format d'affichage ligne 15. C'est pour avoir un affichage au format exponentiel, ainsi 10^{15} s'écrit 1e15. On aurait aussi pu utiliser le format **%g** qui affiche "au mieux".

Cela est-il suffisant ?

Entrez N : 35
35 ! = 2.952328E+38

Entrez N : 36
36 ! = INF

Pour le processeur, au delà de 1e38, c'est l'infini (INF) avec des nombres de type float. On peut aller encore plus loin avec les nombres réel double précision (double) :

```
...  
5  double fact;  
...  
15      printf("%d ! = %g \n", N, fact );
```

Cette fois-ci il faut aller jusqu'à :

Entrez N : 171
171 ! = 7.25742e+306

Entrez N : 172
172 ! = inf

factorielle 172.

Pour information, la fonction factorielle est la fonction qui a le plus fort taux de croissance, $\exp(172)$ ne vaut "que" $4.9963e+74$

On peut encore aller plus loin....

...

```
5 long double fact;
```

...

```
15 printf("%d ! = %LE \n", N, fact );
```

Entrez N : 1755

1755 ! = 1.979262E+4930

Entrez N : 1756

1756 ! = INF

On peut mener les calculs jusqu'à 10^{4930} . Ce qui est suffisant dans l'immense majorité des cas. Noter le **L majuscule** pour le format long dans le printf ligne 15 : %LE

Il ne sert à rien de coder les nombres en double ou quadruple (= long double) précision si on les affiche en simple précision !