Somme de nombres [lp02]
Exercice résolus

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 17 mai 2018

Table des matières

C++ - Somme de nombres (Solution)

Utilise Structures répétitives ■
Durée estimée 1 h ■

Objectif
Cet exercice calcule la somme d'une série de nombres (ici des entiers) donnés par l'utilisateur. Pour qu'il soit **profitable**, essayez de le faire par vous même avant de le visualiser et/ou de télécharger les solutions.

1 Somme de nombres / pgsomme

On veut calculer la somme d'une série d'entiers. Il faut d'abord se demander comment l'utilisateur va pouvoir indiquer combien de nombres il faut additionner, ou bien quand est-ce que le dernier nombre à additionner a été entré. Voyons quelques possibilités.

1.1 Variante 1

Au départ l'utilisateur indique le nombre `n` de termes. Ce problème est proche de ce qui a été fait dans @[Afficher les entiers pairs].

Écrivez un programme qui saisit un entier dans `n`.
Affichez l'invite :

```
n?
```

On souhaite sommer. Il convient donc de,
Déclarez une variable `somme` (ou `s`) (entier) et de l'initialiser à zéro (car c'est une somme).

Écrivez ensuite une répétitive `Itérer` ou `Pour` qui varie de 1 à `n` (inclus) et dans laquelle, saisissez un entier dans `nombre` (ou `nb`) que l'on ajoute ensuite à la `somme`.

Finalement affichez (où `[x]` désigne le contenu de `x`) :

```
Somme = [somme]
```

Testez. Exemple d'exécution :

```
n? 6
5 -2 3 7 0 12
Somme = 25
```

Validez votre programme avec la solution.

Solution C++ @[pgsomme1.cpp]

```cpp
#include <iostream>
using namespace std;

int main()
{
  int n;
  cout<<"n? ";
  cin>>n;
  int somme = 0;
```

```
  for (int j = 1; j <= n; ++j)
  {
    int nombre;
    cin>>nombre;
    somme += nombre;
  }
  cout<<"Somme = "<<somme<<endl;
}
```

1.2 Variante 2

Après chaque nombre, on demande à l'utilisateur s'il y a encore un nombre à additionner. Ici, il faut donc chercher une solution différente car on ne connait pas au départ le nombre de valeurs à additionner et donc le nombre d'exécution de la boucle. On va devoir passer à une répétitive inconditionnelle (`TantQue` ou `Répéter`). Si on envisage de demander en fin de boucle s'il reste encore un nombre à additionner, cela donne :

Au choix :

- Écrivez un nouvel programme,
- Mettez la boucle `Pour` en commentaire et modifiez votre programme,

de sorte qu'il effectue la somme d'entiers en utilisant une structure `TantQue` ou `Répéter`. A chaque fois, demandez si l'utilisateur souhaite finir ou continuer.

Testez. Exemple d'exécution :

```
5 true -2 true -3 true 8 true 6 false
Somme = 14
```

Validez votre programme avec la solution.

Solution C++, TantQue @[pgsomme2a.cpp]

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int somme = 0;
  bool encore = true;
  while (encore)
  {
    int nombre;
    cin>>nombre;
    somme += nombre;
    string rep;
    cin>>rep;
    encore = (rep == "true");
```

```
  }
  cout<<"Somme = "<<somme<<endl;
}
```

Solution C++, Répéter @[pgsomme2b.cpp]

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int somme = 0;
  bool encore;
  do {
    int nombre;
    cin>>nombre;
    somme += nombre;
    string rep;
    encore = (rep == "true");
  } while (encore);
  cout<<"Somme = "<<somme<<endl;
}
```

1.3 Variante 3

L'utilisateur entre une valeur spéciale pour indiquer la fin. On parle de valeur **sentinelle**. Ceci n'est possible que si cette valeur **sentinelle** ne peut pas être un terme valide de l'addition. Par exemple, si on veut additionner des nombres positifs uniquement, la valeur -1 peut servir de valeur sentinelle. Mais sans limite sur les nombres à additionner (positifs, négatifs ou nuls), il n'est pas possible de choisir une sentinelle.

Comme nous souhaitons additionner des entiers et qu'un zéro ne modifie pas le calcul, nous prendrons la valeur 0 pour la sentinelle.

Au choix :

- Écrivez un nouvel programme,
- Modifiez votre programme,

de sorte qu'il définit une constante `SENTINELLE=0` puis, `TantQue` l'entier lu n'est pas la `SENTINELLE`, cumule sa valeur dans la `somme` puis saisit l'entier suivant.

Testez. Exemple d'exécution :

```
5 3 -2 7 -1 11 9 9 0
Somme = 41
```

Validez votre programme avec la solution.

Solution C++ @[pgsomme3.cpp]

```cpp
#include <iostream>
using namespace std;
const int SENTINELLE = 0;

int main()
{
  int somme = 0;
  int nombre;
  cin>>nombre;
  while (nombre != SENTINELLE)
  {
    somme += nombre;
    cin>>nombre;
  }
  cout<<"Somme = "<<somme<<endl;
}
```

Solution commentée

Ici, on se base sur la valeur entrée pour décider si on continue ou pas. Il faut donc **toujours** effectuer un test après une lecture de valeur. C'est pour cela qu'il faut effectuer une lecture avant et une autre à la fin de la boucle.

Quelle valeur sentinelle prendrait-on pour additionner :

1. Une série de notes d'interrogations ?
2. Une série de températures ?

2 Que retenir de cet exercice ?

Initialiser une variable signifie lui donner une première valeur, soit par une affection, soit par une saisie.

Il y a deux modèles de répétitives qui s'appliquent chacun à une situation différente :

- Nombre de répétitions connu avant d'acquérir les valeurs et les traiter :

```
obtenir le nombre de valeurs (nbval)
Pour j <- 1 à nbval Faire
|  obtenir la valeur
|  traiter la valeur
FinPour
```

- Nombre de répétitions inconnu mais arrêt de la saisie marqué par une valeur fictive dite *sentinelle* qui ne doit pas être traitée :

```
obtenir la première valeur
TantQue la valeur n'est pas celle de la sentinelle Faire
|  traiter la valeur
|  obtenir la valeur suivante
FinPour
```

Il faut prendre conscience qu'avec les répétitives, ordre d'écriture et ordre chronologique ne concordent plus. En effet, dans la boucle `TantQue`, une valeur saisie pendant le tour `n` sera traitée au tour `n+1`.

3 Références générales

Comprend [Chaty-PG1 :c3 :ex1] ■