

Autour de la somme [rc02] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  Version 21 mai 2018

Table des matières

1	Autour de la somme / pgsomme	2
1.1	Somme récursive	2
1.2	Somme récursive des pairs	5
2	Références générales	5

C++ - Autour de la somme (Solution)



Mots-Clés Récursivité des actions ■

Requis Algorithmes paramétrés ■

Difficulté ●○○ (20 min) ■



Objectif

Cet exercice calcule des sommes récursivement.

1 Autour de la somme / pgsomme

1.1 Somme récursive



Écrivez une fonction récursive `somme(n)` qui calcule et renvoie la somme des entiers compris entre 1 et `n` (entier).



Validez votre fonction avec la solution.

Solution C++ @[pgsomme.cpp]

```
/**
 * Somme récursive
 * @param[in] n - un entier
 * @return Somme des entiers de 1 à n
 */
int somme(int n)
{
    return (n <= 0 ? 0 : n + somme(n - 1));
}
```



Explicitez le calcul de `somme(3)`.

Solution simple

Si l'on s'en tient à la définition de la fonction, cela conduit à effectuer la suite des opérations suivantes :

```
somme(3)
3 + somme(2)
3 + (2 + somme(1))
3 + (2 + (1 + somme(0)))
3 + (2 + (1 + 0))
3 + (2 + 1)
3 + 3
6
```

L'appel de `somme(3)` nécessite celui de `somme(2)`, qui lui-même nécessite `somme(1)`, qui appelle à son tour jusqu'à aboutir à `somme(0)`. Ce dernier appel est le premier qui renvoie un résultat (en l'occurrence 0). Cette première phase est la *descente* dans la récursion. Cela permet alors de calculer `somme(1)`, puis `somme(2)` et enfin `somme(3)` qui vaut 6 : cette deuxième phase est la *remontée* dans la récursion.



Écrivez une fonction récursive `sommetr(n)` qui « trace » le calcul de la somme des entiers compris entre 1 et `n` (entier). Voici le résultat d'exécution pour `\lstinlinen=3@`.

```
--> sommetr(3)
--> sommetr(2)
--> sommetr(1)
--> sommetr(0)
<-- sommetr(0)=0
<-- sommetr(1)=1
<-- sommetr(2)=3
<-- sommetr(3)=6
==> sommetr(3) vaut 6
```

Les appels à la fonction `somme` sont repérés par le signe `-->` et les valeurs retournées sont repérées par le signe `<--`.



Validez votre fonction avec la solution.

Solution C++ @[pgsomme.cpp]

```
/**
 * Trace de la Somme récursive
 * @param[in] n - un entier
 * @return Somme des entiers de 1 à n
 */
int sommetr(int n)
{
    int rs = 0;
    cout<<"--> sommetr("<<n<<")"<<endl;
    if (n > 0)
    {
        rs = n + sommetr(n - 1);
    }
    cout<<"<-- sommetr("<<n<<")"<<endl;
    return rs;
}
```



Écrivez une fonction récursive terminale `sommeRt(n,a)` qui calcule et renvoie la somme des entiers compris entre 1 et `n` (entier), l'entier `a` étant le résultat.



Validez votre fonction avec la solution.

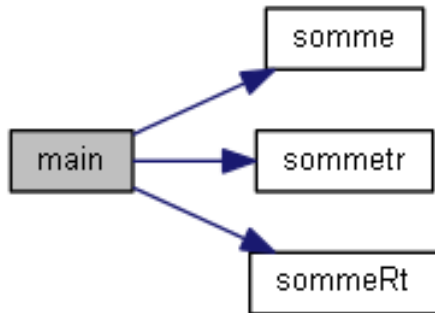
Solution C++ @[pgsomme.cpp]

```
/**
 * Somme récursive terminale
 * @param[in] n - un entier
 * @param[in] a - cumul
 * @return Somme des entiers de 1 à n
 */
int sommeRt(int n, int a)
```

```
{
    return (n <= 0 ? a : sommeRt(n - 1, n + a));
}
```



Écrivez une procédure `test1` qui saisit un entier puis appelle et affiche le résultat des diverses fonctions.



Testez. Exemple d'exécution :

```
Somme de 1 à ? 3
==> somme(3) vaut 6
--> sometr(3)
--> sometr(2)
--> sometr(1)
--> sometr(0)
<-- sometr(0)=0
<-- sometr(1)=1
<-- sometr(2)=3
<-- sometr(3)=6
==> sometr(3) vaut 6
==> sommeRt(3) vaut 6
```



Validez votre procédure avec la solution.

Solution C++

@[pgsomme.cpp]

```
int main()
{
    int n;
    cout<<"Somme de 1 a ? ";
    cin>>n;
    cout<<"==> somme(n) vaut "<< somme(n)<<endl;
    cout<<"==> sometr(n) vaut "<< sometr(n)<<endl;
    cout<<"==> sommeRt(n) vaut "<< sommeRt(n,0)<<endl;
}
```

1.2 Somme récursive des pairs



Écrivez une fonction `sommePairs(x,n)` qui réalise la somme des `n` nombres naturels pairs consécutifs, à partir d'une valeur paire donnée `x` (la valeur de départ sera comprise dans la somme).

Solution simple

Il convient de définir une fonction récursive `sommePairs(x,i,nb)` qui réalise la somme des `nb` nombres naturels pairs consécutifs, à partir d'une valeur paire donnée `x` en ayant déjà additionné `i` nombres. Le cas d'arrêt se présente quand le nombre de valeurs déjà additionnées correspond au nombre total prévu ; et dans ce cas, le résultat est l'élément neutre de la l'addition : 0. Dans les autres cas, l'appel récursif permet de calculer la somme des autres valeurs paires restantes, à laquelle se rajoute la valeur du nombre courant.



Écrivez une procédure `test2` qui saisit deux entiers puis appelle et affiche le résultat de la fonction.



Testez.



Validez votre fonction avec la solution.

Solution Python

```
def sommeNatPairRec(x, i, nb):
    if i == nb:
        somme = 0
    else:
        somme = x + sommeNatPairRec(x+2, i+1, nb)
    return somme

def sommeNatPair(x, n):
    return sommeNatPairRec(x, 0, n)

x = int(input("Donner l'entier de départ (pair) :"))
if x%2 != 0: x += 1 # ajuster au premier pair supérieur
n = int(input("Combien de valeurs à sommer : "))
print("Somme de", n, "naturels pairs à partir de", x, ":", sommeNatPair(x, n))
```

2 Références générales

Comprend [Chappelier-CPP1 :c06 :et], [Felea-PG1 :c3 :ex84] ■