TD1 : Nombres parfaits

1 Le problème

Un nombre entier $N > 1$ est dit **parfait** s'il est égal à la somme de tous ses diviseurs autres que lui même. Par exemple :
– Le nombre 6 est parfait car ses diviseurs sont 1, 2, 3 et 6 et on a $6 = 1 + 2 + 3$.
– Le nombre 10 n'est pas parfait car ses diviseurs sont 1, 2, 5 et 10 et $10 \neq 1 + 2 + 5$.
– Le nombre 28 est parfait car ses diviseurs sont 1, 2, 4, 7, 14 et 28 et on a $28 = 1 + 2 + 4 + 7 + 14$.
Le but de ce TP est de déterminer le seul nombre parfait de trois chiffres et calculer la somme des inverses de ses diviseurs.

2 Les questions

1. Quel est le rôle de la fonction `li(n)` suivante ?

```
Fonction li(n : entier) : liste
Rôle : ...................
Variables locales : L (liste), k (entier)
Début
   L <- []
   Pour k de 1 à n Faire
      Si n mod k = 0 alors
       Ajouter k à L
      FinSi
   FinPour
   Retourner(L)
FinFonction
```

2. Écrire une fonction booléenne `estParfait(n)` qui renvoie VRAI si le nombre entier n est parfait et FAUX sinon.
   Par exemple, on devrai avoir `estParfait(6) = VRAI` et `estParfait(7) = FAUX`.
3. Écrire une fonction `sominvdiv(n)` qui renvoie la somme des inverses des diviseurs d'un nombre entier n non nul.
   Par exemple, puisque les diviseurs de 10 sont 1, 2, 5 et 10, on devra avoir
   `sominvdiv(10)` $= \frac{1}{1} + \frac{1}{2} + \frac{1}{5} + \frac{1}{10} = 1 + 0,5 + 0,2 + 0,1 = 1,8$
4. Écrire un algorithme déterminant et affichant le seul nombre parfait de trois chiffres puis la somme des inverses des diviseurs de ce nombre. Cet algorithme devra faire appel aux fonctions précédentes.
   Implémenter en C++ cet algorithme puis exécuter le programme.
5. Question subsidiaire : à votre avis, dans cet algorithme, combien de tests ont été faits ?
   Est-ce optimal ?
   Comment pourrait-on rendre le programme plus rapide ?

3 Pour l'implémentation en C++

Vous pourrez utiliser la bibliothèque `list` en rajoutant en début de fichier :

```
#include <list>
```

Pour déclarer une liste :

```
list<int> L;
```

Pour ajouter un élément en tête de liste :

```
L.push_front(k);
```

Pour accéder au premier élément de la liste :

```
L.front();
```

Pour décapiter la tête de liste (retirer le premier élément) :

```
L.pop_front();
```