**IREM d'Aquitaine**
Institut de Recherche sur l'Enseignement des Mathématiques
40, rue Lamartine, 33400 Talence

# Nombres parfaits

**Fiche élève**

On souhaite produire un algorithme permettant de déterminer si un nombre entier positif non nul est *parfait* ou non.

## Travail préparatoire

1. Donner la liste des diviseurs de 36 et la liste des diviseurs de 60.

- Diviseurs de 36 : .....
- Diviseurs de 60 : .....

Considérons l'algorithme suivant, au format Algobox :

```
****************************************
Que fait cet algorithme ?...
****************************************
1   VARIABLES
2     n EST_DU_TYPE NOMBRE
3     i EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     LIRE n
6     POUR i ALLANT_DE 1 A n
7       DEBUT_POUR
8       SI ( n % i == 0 ) ALORS
9         DEBUT_SI
10        AFFICHER i
11        AFFICHER " "
12        FIN_SI
13      FIN_POUR
14  FIN_ALGORITHME
```

***Note.*** *L'opérateur* `%` *permet d'obtenir le reste de la division entière de deux nombres.*

2. Créer cet algorithme sous Algobox et le tester. Que fait cet algorithme ?

## Conception de l'algorithme

3. Modifier l'algorithme précédent afin qu'il calcule puis affiche la somme des diviseurs d'un nombre.

> On dit qu'un nombre entier positif non nul est *parfait* s'il est égal à la somme de ses diviseurs stricts (c'est-à-dire autres que lui même).

4. Modifier l'algorithme précédent afin qu'il permette de déterminer si un nombre est parfait ou non.

5. Tester cet algorithme avec 28, puis avec 36 et 60.

6. Quel est le plus petit entier naturel parfait ?

## Extension

> On dit que deux nombres entiers positifs non nuls sont *amiables* si la somme des diviseurs stricts de chaque nombre est égale à l'autre nombre.

7. Vérifier que 220 et 284 sont amiables.

8. Modifier l'algorithme précédent afin qu'il permette de déterminer si deux nombres entiers positifs non nuls sont amiables ou non.

**IREM d'Aquitaine**
Institut de Recherche sur l'Enseignement des Mathématiques
40, rue Lamartine, 33400 Talence

# Nombres parfaits

**Fiche enseignant**

**Objectifs.** Introduction à la structure répétitive `POUR`. Utilisation d'Algobox.

**Prérequis.** Notions de base d'algorithmique (notion de variable, entrées-sorties, structure conditionnelle `SI-ALORS-SINON`).

## Travail préparatoire

1. Donnez la liste des diviseurs de 36 et la liste des diviseurs de 60.

- Diviseurs de 36 : 1, 2, 3, 4, 6, 9, 12, 18, 36.....
- Diviseurs de 60 : 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60.....

Considérons l'algorithme suivant, au format Algobox.

```
****************************************
Que fait cet algorithme ?...
****************************************
1   VARIABLES
2     n EST_DU_TYPE NOMBRE
3     i EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     LIRE n
6     POUR i ALLANT_DE 1 A n
7       DEBUT_POUR
8       SI ( n % i == 0 ) ALORS
9         DEBUT_SI
10        AFFICHER i
11        AFFICHER " "
12        FIN_SI
13      FIN_POUR
14  FIN_ALGORITHME
```

***Note.*** *L'opérateur* `%` *permet d'obtenir le reste de la division entière de deux nombres.*

2. Créer cet algorithme sous Algobox et le tester. Que fait cet algorithme ?

*On observe le résultat et on « lit » la structure de l'algorithme pour comprendre…*

Cet algorithme affiche la liste des diviseurs d'un nombre.

*La structure* `POUR` *permet de répéter l'exécution d'un bloc en faisant varier automatiquement une variable, dite « variable de boucle ». Dans cet exemple, la variable* `i` *varie automatiquement de 1 à* `n`.

*Ainsi, pour chaque valeur de* `i`*, on affiche la valeur de* `i` *(suivi d'un espace pour éviter que les valeurs affichées soient « collées » les unes aux autres) lorsque* `i` *divise* `n`*.*

## Conception de l'algorithme

3. Modifier l'algorithme précédent afin qu'il calcule puis affiche la somme des diviseurs d'un nombre.

*Il est nécessaire d'utiliser une variable supplémentaire pour calculer la somme des diviseurs. Cette variable doit être initialisée à 0, puis être incrémentée à chaque nouveau diviseur découvert. L'affichage du résultat doit désormais se faire à la fin du calcul, et donc en dehors de la structure* `POUR`*. On obtient l'algorithme suivant :*

```
SommeDiviseurs  -  26.01.2015
****************************************
Cet algorithme calcul et affiche la somme des diviseurs d'un entier positif
non nul.
****************************************
1   VARIABLES
2     n EST_DU_TYPE NOMBRE
3     i EST_DU_TYPE NOMBRE
4     somme EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE n
7     somme PREND_LA_VALEUR 0
8     POUR i ALLANT_DE 1 A n
9       DEBUT_POUR
10      SI (n % i == 0) ALORS
11        DEBUT_SI
12        somme PREND_LA_VALEUR somme + i
13        FIN_SI
14      FIN_POUR
15    AFFICHER "La somme des diviseurs de "
16    AFFICHER n
17    AFFICHER " vaut "
18    AFFICHER somme
19  FIN_ALGORITHME
```

On dit qu'un nombre entier positif non nul est ***parfait*** s'il est égal à la somme de ses diviseurs stricts (c'est-à-dire autres que lui même).

4. Modifier l'algorithme précédent afin qu'il permette de déterminer si un nombre est parfait ou non.

*Il faut cette fois calculer la somme des diviseurs stricts de n. Pour cela, il suffit de modifier l'intervalle parcouru par la structure* `POUR` *ainsi :*

```
8     POUR i ALLANT_DE 1 A n / 2
```

*Enfin, on modifie l'affichage du résultat ainsi :*

```
15    AFFICHER "Le nombre "
16    AFFICHER n
17    SI (n == somme) ALORS
```

```
18      DEBUT_SI
19      AFFICHER " est parfait."
20      FIN_SI
21      SINON
22        DEBUT_SINON
23        AFFICHER " n'est pas parfait (la somme de ses diviseurs stricts
vaut "
24        AFFICHER somme
25        AFFICHER ")."
26        FIN_SINON
```

5. Tester cet algorithme avec 28, puis avec 36 et 60.

*28 est parfait, 36 et 60 ne le sont pas… Un autre nombre parfait est, par exemple, 496.*

6. Quel est le plus petit entier naturel parfait ?

*On teste les entiers successivement, et on découvre que le premier entier parfait et 6 = 1 + 2 + 3.*

## Extension

On dit que deux nombres entiers positifs non nuls sont *amiables* si la somme des diviseurs stricts de chaque nombre est égale à l'autre nombre.

7. Vérifier que 220 et 284 sont amiables.

*On utilise l'algorithme précédent et on constate que c'est effectivement le cas (naturellement, ni 220 ni 284 n'est parfait)… Autre exemple : 1184 et 1201.*

8. Modifier l'algorithme précédent afin qu'il permette de déterminer si deux nombres entiers positifs non nuls sont amiables ou non.

*Il suffit de « dupliquer » le calcul précédent : somme1 pour la somme des diviseurs de n1, somme2 pour la somme des diviseurs de n2, puis faire le test et afficher le résultat… On obtient ainsi :*

```
EntiersAmiables  -  26.01.2015
****************************************
Cet algorithme détermine si deux entiers positifs non nuls sont amiables ou
non.
****************************************
1   VARIABLES
2     i EST_DU_TYPE NOMBRE
3     n1 EST_DU_TYPE NOMBRE
4     somme1 EST_DU_TYPE NOMBRE
5     n2 EST_DU_TYPE NOMBRE
6     somme2 EST_DU_TYPE NOMBRE
7   DEBUT_ALGORITHME
8     AFFICHER "Entrez les deux nombres"
9     LIRE n1
10    LIRE n2
11    somme1 PREND_LA_VALEUR 0
12    POUR i ALLANT_DE 1 A  n1 / 2
13      DEBUT_POUR
14      SI (n1 % i == 0) ALORS
15        DEBUT_SI
```

```
16        somme1 PREND_LA_VALEUR somme1 + i
17        FIN_SI
18      FIN_POUR
19    somme2 PREND_LA_VALEUR 0
20    POUR i ALLANT_DE 1 A  n2 / 2
21      DEBUT_POUR
22      SI (n2 % i == 0) ALORS
23        DEBUT_SI
24        somme2 PREND_LA_VALEUR somme2 + i
25        FIN_SI
26      FIN_POUR
27    AFFICHER "Les nombres "
28    AFFICHER n1
29    AFFICHER " et "
30    AFFICHER n2
31    SI ((n1 == somme2) ET (n2 == somme1)) ALORS
32      DEBUT_SI
33      AFFICHER " sont amiables."
34      FIN_SI
35      SINON
36        DEBUT_SINON
37        AFFICHER " ne sont pas amiables (les sommes valent "
38        AFFICHER somme1
39        AFFICHER " et "
40        AFFICHER somme2
41        AFFICHER ")."
42        FIN_SINON
43  FIN_ALGORITHME
```

## Remarque

On peut naturellement remarquer qu'il n'est pas nécessaire d'aller au-delà de n / 2 pour lister les diviseurs d'un nombre… On pourrait ainsi réécrire le premier algorithme de la façon suivante (sans oublier d'afficher le diviseur n) :

```
4   DEBUT_ALGORITHME
5     LIRE n
6     POUR i ALLANT_DE 1 A n / 2
7       DEBUT_POUR
8       SI ( n % i == 0 ) ALORS
9         DEBUT_SI
10        AFFICHER i
11        AFFICHER " "
12        FIN_SI
13      FIN_POUR
14    AFFICHER n
15  FIN_ALGORITHME
```