Sous-programmes

Sous-programmes

Fraction irréductible

En mathématiques, une **fraction est irréductible** s'il n'existe pas de fraction égale ayant des termes plus petits. Autrement dit, une fraction irréductible **ne peut pas être simplifiée**.

Théorème

Soient **a** un entier et **b** un entier naturel non nul. Alors $\frac{a}{b}$ est irréductible si et seulement si **a** et **b** sont premiers entre eux.

Exemple

La fraction $\frac{12}{20}$ n'est pas irréductible car 12 et 20 sont des multiples de 4 : $\frac{12}{20} = \frac{3 \times 4}{5 \times 4} = \frac{3}{5}$ (simplification par 4). On peut aussi écrire $\frac{12}{20} = \frac{12:4}{20:4} = \frac{3}{5}$.

La fraction $\frac{3}{5}$ est irréductible car 1 est le seul entier positif qui divise à la fois 3 et 5.

Méthode de simplification

Pour réduire directement une fraction, il suffit de **diviser le numérateur et le dénominateur par leur plus grand commun diviseur**. D'après le lemme de Gauss, cette forme réduite est unique.

Exemple

Pour réduire la fraction $\frac{42}{390}$, on calcule $\mathrm{PGCD}(42, 390) = 6$ puis on simplifie par 6 : $\frac{42}{390} = \frac{6 \times 7}{6 \times 65} = \frac{7}{65}$.

Problème

On souhaite écrire un programme qui calcule la somme de deux fractions :

$$\frac{p1}{q1} + \frac{p2}{q2} = \frac{ps}{qs}$$

Figure 1, Somme de deux fractions avec : p1, p2, ps ∈ ℤ et q1, q2, qs ∈ ℤ*

Pour résoudre ce problème on propose l'algorithme suivant :

Algorithme

```
Algorithme Somme_Fraction
Début
  // Saisie des deux fractions
  saisie_fraction(p1, q1)
  saisie_fraction(p2, q2)
  // Simplifier les deux fractions
  simplifier_fraction(p1, q1)
  simplifier_fraction(p2, q2)
  // Calculer la somme des deux fractions
  // Puis la simplifier
  somme_fraction(p1, q1, p2, q2, ps, qs)
  // Afficher le résultat
  Ecrire(p1, "/", q1, "+", p2, "/", q2, "=", ps, "/", qs)
Fin
```

Objet	Type/Nature
p1, q1, p2, q2, ps, qs	entier
saisie_fraction simplifier somme_fraction	procédure

Ce programme qui semble, au début, compliqué est devenu assez simple grâce à la **décomposition modulaire**.

Décomposition modulaire

L'**analyse modulaire**, appelée également **décomposition modulaire**, consiste à diviser un problème en sous problèmes de difficultés moindres.

En algorithmique, les **sous problèmes** correspondent à des **sous-programmes**.

Sous-programme

Un **sous-programme** est une section de code nommée ***qui peut être appelée*** en écrivant le nom du sous-programme dans une instruction du programme.

Les **sous-programmes** sont également appelés **procédures** ou **fonctions**.

Une **procédure** exécute simplement un ensemble d'instructions, tandis qu'une **fonction** renvoie une valeur une fois son exécution est terminée.

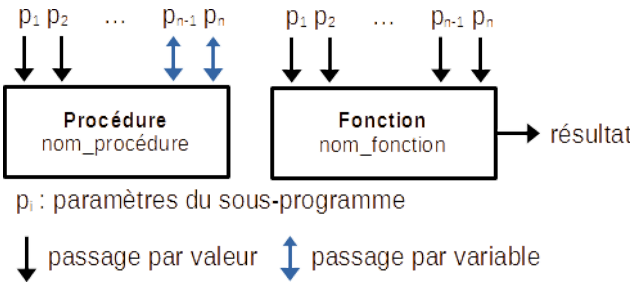

Figure 2, Différence entre une procédure et une fonction

L'écriture de **sous-programmes** rend le code plus lisible et réutilisable, car le code est subdivisé en des sections plus petites.
La plupart des langages de programmation sont livrés avec un ensemble de sous-programmes intégrés, mais permettent, aussi, au programmeur d'écrire leurs propres sous-programmes personnalisés.

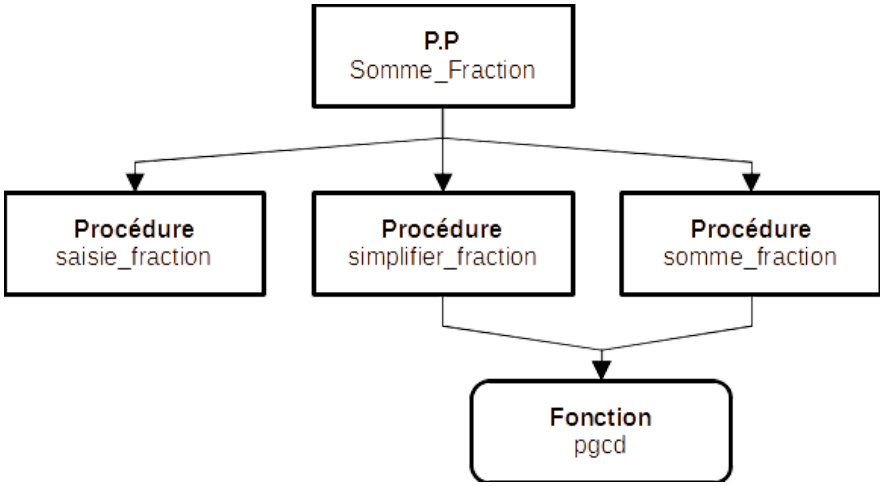

Figure 3, Décomposition modulaire du problème de la somme des deux fractions

Fonction

Définition

Une **fonction** est un sous-programme **qui retourne à son appelant un seul résultat** *en fonction de ses paramètres.*
Une **fonction** peut avoir zéro ou plusieurs paramètres. Ces **paramètres** sont ***souvent transmis par valeur***.

Appel

Comme une **fonction renvoie toujours une valeur**, son appel peut se faire de différentes manières :

- Dans une affectation :

```
Algorithme
// pgcd(a, b) renvoie le PGCD des deux valeurs
dc ← pgcd(a, b)
```

- Dans une structure conditionnelle :

```
Algorithme
// Afficher si un nombre est premier
Si premier(n) Alors
  Ecrire(n, "est premier")
Sinon
  Ecrire(n, "n'est pas premier")
Fin Si
```

- Dans une structure itérative :

Algorithme

```
// f(x) est une fonction qui admet un extrémum
// en x0 ∈ [0, +∞[
// Recherche de l'extrémum de f(x)
x0 ← 0
TantQue (f(x0+pas) > f(x0)) Faire
  x0 ← x0 + pas
Fin TantQue
```

Algorithme

```
// Saisir une chaine alphabétique
// est_alphabetique(ch) : retrourne Si
// Si ch[i] ∈ ["A", "Z"] U ["a", "z"]
Répéter
  Ecrire("Une chaine alphabétique ? ")
  Lire(ch)
Jusqu'à est_alphabetique(ch)
```

Algorithme

```
// Saisie d'un de valeurs distinctes
// entre les indices n1 et n2
// Les fonctions :
// - min(a, b) : renvoie le minimum entre a et b
// - max(a, b) : renvoie le maximum entre a et b
// - existe(v, t, n) : recherche l'existence de v dans les n premières cases de t
Pour i de min(n1, n2) à max(n1, n2)-1 Faire
  Répéter
    Ecrire("t[", i, "] ? ")
    Lire(t[i])
  Jusqu'à (non existe(t[i], t, i-1))
Fin TantQue
```

- Comme paramètre d'un autre sous-programme :

Algorithme

```
// somme_carre(a, b) renvoie a² + b²
Ecrire(somme_carre(a, b))
```

Algorithme

```
// calculer PGCD de a, b et c
dc ← pgcd(a, pgcd(b, c))
```

Vocabulaire et Syntaxe

Algorithme

```
Fonction NomFonction(p1: type1, p2: type2, ...,
                     pn: typen):typerésultat
  //
  // Traitements
  //
  Retourner résultat
Fin
```

Python

```
def NomFonction(p1, p2, ..., pn):

    #
    # Traitements
    #
    return résultat
```

Procédure

Définition

Une **procédure** est un sous-programme **qui ne retourne pas, directement, de résultats à son appelant**.
Une **procédure** peut avoir zéro ou plusieurs paramètres. Ces **paramètres** peuvent être, selon le besoin, ***transmis par valeur ou par variable***.
En algorithmique, le **mode de passage par variable** *est utilisé pour renvoyer, indirectement, un ou plusieurs résultats à l'appelant*. On dit qu'**elle possède un effet de bord**.

Appel

Comme une **procédure ne renvoie aucune valeur**, son appel se fait toujours de la même façon :

Algorithme

```
// Saisir une valeur dans n
saisir(n)
// Remplir le tableau t par n valeurs distinctes
remplir_tab(t, n)
// Echanger le contenu de deux variables
permuter(a, b)
```

Une **procédure** utilise les paramètres passés par valeur pour réaliser ses traitements. Elle peut, aussi, modifier la valeur des paramètres transmis par variable, directement, chez l'appelant.

Vocabulaire & Syntaxe

Algorithme

```
Procédure NomProcédure(p1: type1, p2: type2, ..., pn: typen)
  //
  // Traitements
  //
Fin
```

Python

```
def NomProcedure(p1, ..., pn):
    #
    # Traitements
    #
```

Solution

Procédure saisie_fraction

Algorithme

```
procédure saisie_fraction(@num, @denom: entier)
Début
  Ecrire("Numérateur ? "); Lire(num)
  Répéter
    Ecrire("Dénominateur ≠ 0 ? "); Lire(denom)
  Jusqu'à denom ≠ 0
Fin
```

Python

```
def saisie_fraction():
    p = int(input("Numérateur ? "))
    q = 0
    while not (q != 0):
        q = int(input("Dénominateur ≠ 0 ? "))
    return p, q
```

Procédure simplifier_fraction

Algorithme

```
procédure simplifier_fraction(@num, @denom: entier)
Début
  dc ← pgcd(num, denom)
  num ← num div dc
  denom ← denom div dc
Fin
```

Python

```
def simplifier_fraction(p, q):
    dc = pgcd(p, q)
    p = p // dc
    q = q // dc
    return p, q
```

Procédure somme_fraction

Algorithme

```
procédure simplifier_fraction(p1, q1, p2, q2: entier;
                              @num, @denom: entier)
Début
  num ← p1 * q2 + p2 * q1
  denom ← q1 * q2
  simplifier_fraction(num, denom)
Fin
```

Python

```
def somme_fraction(p1, q1, p2, q2):
    p = p1 * q2 + p2 * q1
    q = q1 * q2
    return simplifier_fraction(p, q)
```

Fonction pgcd

Algorithme

```
Fonction pgcd(a, b: entier):entier
Début
  TantQue b ≠ 0 Faire
    r ← a mod b
    a ← b
    b ← r
  Fin TantQue
  retourner a
Fin
```

Python

```
def pgcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

Programme Principal

Algorithme

```
Algorithme Somme_Fraction
Début
  // Saisie des deux fractions
  saisie_fraction(p1, q1)
  saisie_fraction(p2, q2)
  // Simplifier les deux fractions
  simplifier_fraction(p1, q1)
  simplifier_fraction(p2, q2)
  // Calculer la somme des deux fractions
  // Puis la simplifier
  somme_fraction(p1, q1, p2, q2, ps, qs)
  // Afficher le résultat
  Ecrire(p1, "/", q1, "+", p2, "/", q2, "=", ps, "/", qs)
Fin
```

Python

```
# PP
p1, q1 = saisie_fraction()
p2, q2 = saisie_fraction()

p1, q1 = simplifier_fraction(p1, q1)
p2, q2 = simplifier_fraction(p2, q2)
ps, qs = somme_fraction(p1, q1, p2, q2)

print(p1, "/", q1, "+", p2, "/", q2, "=", ps, "/", qs)
```