Afin de pouvoir compter le nombre d'opérations effectuées par un algorithme, il faut tout d'abord savoir ce qu'on appelle **UNE opération**. Est considérée comme une opération le fait de :

- utiliser une opération mathématique de base (addition, soustraction, multiplication, division,...),
- utiliser la valeur d'une variable (il faut aller chercher cette valeur en mémoire),
- donner une nouvelle valeur à une variable (il faut aller écrire cette valeur en mémoire),
- lire une valeur simple (entier, décimal) donnée par l'utilisateur,
- écrire une valeur simple (entier, décimal),
- effectuer un test,
- utiliser une opération booléenne ("et", "ou", "non").

Parmi ces opérations certaines prennent en pratique plus de temps que d'autres mais, afin de simplifier les calculs, nous ferons l'approximation qu'elles sont toutes aussi rapides. Il existe bien entendu d'autres opérations mais celles indiquées ci-dessous sont suffisantes pour le moment. Il vous sera facile d'adapter les cours de ce chapitre à des algorithmes plus complexes.

Prenez une feuille de papier, et faites les exercices ci-dessous l'un après l'autre. Pour chacun d'entre eux, écrivez votre réponse sur votre feuille en prenant le temps de bien y réfléchir. Regardez alors la solution et, si vous n'aviez pas la bonne valeur, essayez de comprendre pourquoi. Passez alors à l'exercice suivant.

Pour chaque exercice, lisez bien l'algorithme proposé et indiquez ce qu'il va faire/calculer/afficher. Indiquez alors pour chaque type d'opération effectuée combien de fois elle est effectuée, ainsi que le nombre d'opérations total. Par exemple : "1 addition, 2 multiplications donc 3 opérations au total".

Exercice 1

```
1 largeur <- LireEntier()
2 longueur <- LireEntier()
3 aire <- largeur * longueur
4 perimetre <- (largeur + longueur) * 2
5 Afficher aire
6 Afficher perimetre
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est de calculer (puis afficher) l'aire et le périmètre d'un rectangle dont on a donné la longueur et la largeur. Il effectue 17 opérations :

- 2 multiplications (lignes 3 et 4) et 1 addition (ligne 4),
- 4 affectations de variables (lignes 1 à 4),
- 6 utilisations de variables (2 aux lignes 3 et 4, 1 aux lignes 5 et 6),
- 2 lectures et 2 écritures.

Exercice 2

```
1 nbLivres <- LireEntier()
2 Si nbLivres < 10
3    prix <- nbLivres * 10
4 Sinon
5    prix <- nbLivres * 9
6 Afficher prix
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est de calculer le prix qu'il faut payer en fonction du nombre de livres achetés. Si on prend moins de 10 livres le prix est de 10 (euros ?) par livre, sinon il est de 9 par livre. Il effectue 9 opérations :

- 1 multiplication (lignes 3 ou 5),
- 2 affectations de variables (lignes 1, puis 3 ou 5),
- 3 utilisations de variables (lignes 2, puis 3 ou 5, puis 6),
- 1 lecture et 1 écriture,
- 1 test (ligne 2).

Il faut faire attention au test : on rentre soit dans le "si" soit dans le "sinon", donc il ne faut pas compter les choses en double.

Exercice 3

```
1 X <- 1
2 Tant que X <= 100
3    Afficher "Bonjour !"
4    X <- X + 1
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est d'afficher 100 fois le texte "Bonjour !". Dans la boucle, il effectue à chaque tour 6 opérations (1 utilisation de variable, 1 test, 1 écriture, 1 utilisation de variable, 1 addition et 1 affectation de variable) et avant la boucle, il effectue juste 1 affectation de variable. Comme la boucle est effectuée 100 fois au total on a donc :

- 100 additions,
- 100 * 2 + 1 = 201 utilisations de variables (le +1 vient du test qui nous fait sortir de la boucle),
- 100 + 1 = 101 affectations de variables,
- 100 écritures,
- 101 tests (100 où la condition est vraie et 1 où elle est fausse).

Ce qui fait donc 603 opérations au total.

Exercice 4

```
1 total <- 0
2 i <- 1
3 Tant que i <= 100
4    total <- total + i
5    i <- i + 1
6 Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est de calculer la somme des nombres de 1 à 100 puis de l'afficher. La boucle est effectuée 100 fois, ce qui donne un total de 906 opérations :

- 100 * 2 = 200 additions,
- 100 * 4 + 1 + 1= 402 utilisations de variables (1 ligne 3, 2 ligne 4, 1 ligne 5, 1 ligne 6, 1 ligne 3 pour le test qui arrête la boucle),
- 100 * 2 + 2 = 202 affectations de variables (2 avant la boucle, 2 dans la boucle),
- 1 écriture,
- 101 tests.

Exercice 5

```
1 iMax <- LireEntier()
2 total <- 0
3 i <- 1
4 Tant que i <= iMax
5    total <- total + i
6    i <- i + 1
7 Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est de calculer la somme des nombres de 1 à *iMax* puis de l'afficher. La différence par rapport à l'exercice précédent est que le nombre d'itérations de la boucle n'est plus fixé, mais dépend de *iMax*. En l'occurence, la boucle sera effectuée *iMax* fois. On a donc :

- *iMax* * 2 = *2iMax* additions,
- *iMax* * 5 + 1 + 2 = *5iMax* + *3* utilisations de variables (2 ligne 4, 2 ligne 5, 1 ligne 6, 1 ligne 7, 2 ligne 4 pour le test qui arrête la boucle),
- *iMax* * 2 + 3 = *2iMax+3* affectations de variables (3 avant la boucle, 2 dans la boucle),
- 1 lecture et 1 écriture,
- *iMax* + 1 tests.

Au total on a donc *10iMax+9* opérations.

Le nombre d'opérations peut donc tout à faire dépendre des données lues par l'algorithme et on obtient une formule avec laquelle on peut calculer le nombre exact d'opérations pour toute valeur de *iMax* souhaitée.

Exercice 6

```
1 iMax <- LireEntier()
2 total <- 0
3 Tant que iMax > 0
4    total <- total + iMax
5    iMax <- iMax - 1
6 Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

Le but de cet algorithme est de calculer la somme des nombres de 1 à *iMax* puis de l'afficher. La différence par rapport à l'exercice précédent est qu'on n'utilise plus de variable intermédiaire *i* mais

qu'on fait décroître la variable *iMax*. A nouveau, la boucle sera effectuée *iMax* fois. On a donc :

- *iMax* additions et *iMax* soustractions,
- *4iMax + 2* utilisations de variables (1 ligne 3, 2 ligne 4, 1 ligne 5, 1 ligne 6, 1 ligne 3 pour le test qui arrête la boucle),
- *2iMax+2* affectations de variables (2 avant la boucle, 2 dans la boucle),
- 1 lecture et 1 écriture,
- *iMax + 1* tests.

Au total on a donc *9iMax+7* opérations.

L'algorithme effectue donc la même chose que celui de l'exercice précédent, mais a besoin d'un peu moins d'opérations.

Exercice 7

```
1  taille <- LireEntier()
2  X <- 0
3  Tant que X < taille
4     Y <- 0
5     Tant que Y < taille
6        Si X = Y
7           Afficher "X"
8        Sinon
9           Afficher "."
10       Y <- Y + 1
11    Retour à la ligne
12    X <- X + 1
```

Afficher/cacher la solution

Solution de l'exercice :

Cet algorithme permet d'afficher une diagonale de "X". Sur un exemple, avec *taille* valant 4 :

```
X...
.X..
..X.
...X
```

Pour calculer le nombre d'opérations, on va tout d'abord regarder les lignes 5 à 10, c'est-à-dire la boucle sur *Y*. Il y a au total :

- *taille* additions (ligne 10),
- *5taille + 2* utilisations de variables (2 ligne 5, 2 ligne 6, 1 ligne 10, 2 ligne 5 pour le test qui arrête la boucle),
- *taille* affectations de variables (ligne 10),
- *taille* écritures (ligne 7 ou 9),
- *2taille + 1* tests (lignes 5 et 6).

Si on enlève les lignes 5 à 10, on va avoir :

- *taille* additions (ligne 12),
- *3taille+2* utilisations de variables (2 ligne 3, 1 ligne 12, 2 ligne 3 pour le test qui arrête la boucle),
- *2taille + 2* affectations de variables (lignes 1, 2, 4 et 12),
- 1 lecture et *taille* écritures (ligne 11),
- *taille + 1* tests (lignes 3).

Au total, comme les lignes 5 à 10 sont répétées *taille* fois on aura

- *taille* + *taille* * *taille* additions,
- *3taille* + *2* + *(5taille* + *2)* * *taille* = *5taille* + *2* + *5taille* * *taille* utilisations de variables,
- *2taille* + *2* + *taille***taille* affectations de variables,
- 1 lecture et *taille* + *taille***taille* écritures,
- *taille* + *1* + *(2taille* + *1)* **taille* tests.

Cela donne donc un total de $10taille^2+11taille+6$ opérations.

Calculer le nombre d'opérations effectuées par un algorithme nous permet donc de le comparer à d'autres algorithmes effectuant la même tâche, comme nous l'avons vu sur l'exemple du calcul d'une somme d'entiers. On peut donc ainsi voir quel algorithme sera plus rapide mais, comme vous avez pu le constater, calculer le nombre exact d'opérations devient rapidemment assez fastidieux ! Nous allons voir comment se passer de ce comptage exact.