```
1    with Ada.Text_IO;
2    use Ada.Text_IO;
3
4    procedure Simple_Types is
5       X      :         Integer := 10;
6       Y      : constant Integer := 20;
7       Result :         Integer := 0;
8    begin
9       Result := X + Y;
10
11      Put_Line ("Résultat = " & Integer'Image (Result));
12   end Simple_Types;
```

**Programme 2: simple_types.adb**

Donc, nous avons déjà vu les lignes 1, 2, 4, 8 et 12. Alors quoi de neuf ? Nous n'avons pas vu les définitions de variables et constantes avant, ce sont les lignes 5, 6 et 7. Ici nous définissons 2 variables, **`X`** et Result qui sont de type entier (Integer) et 1 constante, **`Y`**, également de type entier.

La différence entre une variable et une constante est que vous pouvez assigner une valeur à une variable dans le programme, voir la ligne 9, où nous assignons **`X + Y`** à **`Result`**. En Ada, le symbole := indique que la variable à la gauche prend la valeur de ce qui est à la droite, dans notre cas, c'est 10 + 20 donc 30 qui est assigné à Result. Nous utilisons le mot-clé "constant" avant le type (Integer) pour en faire une constante. Alors que se passe-t-il si vous essayez d'assigner une valeur à **`Y`** dans le programme ? Essayez-le vous-même et regardez ce qui se passe lorsque vous compilez le programme. Il ne compilera pas, vous ne pouvez pas assigner une valeur à une constante après l'avoir déclarée.

À la ligne 11, il y a quelque chose d'étrange **`Integer'Image`**. Qu'est-ce-que ceci ? C'est un attribut de Integer. Voir l'encadré "Fonctionnalités sympas : les attributs" pour plus de détails.

Ligne 11 aussi, nous avons un autre symbole, **`&`**, qui signifie concaténation de chaîne de caractères. Ceci signifie que nous pouvons "ajouter" des chaînes ensemble, la partie à gauche de **`&`** est ajoutée à la partie droite et **`Put_Line`** affiche le tout à l'écran.

## Exercices

**1. Changez la ligne 9 pour chacune des lignes suivantes, compilez et exécutez, quelle est la valeur de Result ?**

```
a) X – Y
b) Y - X
c) X * Y
d) X / Y
e) Y / X
```

**2. Passez du temps à essayer différents nombres, variables et constantes et voyez quel est le résultat sur le terminal.**

### *Fonctionnalités sympas : les types*

Les types permettent au compilateur de vérifier que seules des variables de même type soient utilisées ensemble, par exemple, **`X := Y + 10`**; X, Y et 10 sont des entiers, si X était autre chose, disons un booléen, ce programme n'aurait aucun sens et ne compilerait pas ; le compilateur vous donnerait un message utile pour trouver l'erreur.

## Types numériques

En plus du type Integer, il y a 2 autres types qui sont basés sur le type Integer appelés Natural et Positive ; ceux-ci sont des sous-types de Integer car ils restreignent le champ de valeurs permises pour les variables de ces types.