Le typage

1. Introduction

Les ordinateurs manipulent des nombres. Ils stockent des nombres en mémoire, font transiter des nombres par leurs circuits, et envoient des informations aux autres ordinateurs sous forme de nombres.

La programmation, c'est (entre autres) le fait de dialoguer avec l'ordinateur, afin de lui faire faire les opérations que l'on veut : on traduit les opérations dans un langage compréhensible pour l'ordinateur, et il l'exécute.

Comme c'est un ordinateur, il ne manipule que des nombres, et ces opérations se transforment donc forcément, à un moment ou un autre, en opérations sur des nombres. Quand on veut faire faire à l'ordinateur des additions, multiplications ou autres opérations mathématiques, ce n'est pas surprenant de savoir qu'il manipule des nombres. Mais quand on clique sur un bouton, est-ce que l'on conçoit clairement de quelle manière il va interpréter la chose en tant que nombre ? Non. Ce n'est d'ailleurs pas possible, parce que la manière dont est effectuée la traduction dépend de beaucoup de choses, que ce soit au niveau logiciel (encodage, format utilisé...) ou matériel (processeur, mémoire, etc.).

Les langages de bas niveau actuels, ainsi que les premiers langages de programmation sont très proches de la machine sur laquelle ils s'exécutent : les objets qu'ils manipulent sont, eux aussi, des nombres.

En Assembleur par exemple, la majorité des instructions consiste à modifier de manière arithmétique (en additionnant, multipliant, ou effectuant d'autres opérations de ce genre) le contenu des cases mémoire de l'ordinateur.

2. Types prédéfinis

Tous les langages de programmation offrent des types de base correspondant aux données qui peuvent être traitées directement — à savoir : sans conversion ou formatage préalable — par le processeur. Ces types de base sont souvent :

- Type booléen : valeurs vrai ou faux — ou respectivement 1 ou 0 ;
- Type entier signé ou non signé : valeurs codées sur 8 bits, 16 bits, 32 bits voire 64 bits.
- Les caractères sont parfois assimilés à des entiers codés sur 8 ou 16 bits (exemples : C et Java) ;
- Type réel en virgule flottante.

Les langages permettant un accès direct à la mémoire du système offrent par ailleurs le type pointeur, et un type octet.

Beaucoup proposent également un type prédéfini, string, pour les chaînes de caractères. Les langages de haut niveau peuvent également supporter nativement des types correspondant à d'autres structures de données.

3. Des types pour plus de sûreté

Ces langages conviennent très bien quand il s'agit de manipuler des nombres (ils ont au départ été conçus pour des applications scientifiques), mais assez vite des problèmes se posent.

Mettons que vous ayez écrit une opération qui transforme une chaîne de caractères (une suite de nombres en mémoire) par leurs équivalents minuscules. Si on applique par erreur cette opération non pas sur une phrase, mais sur la liste des chiffres, l'ordinateur traduira cette liste en une liste avec des valeurs loufoques.

Cette erreur est aisément repérable : un humain, si on lui demande de transformer en lettres majuscules une liste d'âges, va comprendre immédiatement qu'il y a quelque chose qui cloche. Pourquoi l'ordinateur ne pourrait-il pas faire pareil ?

C'est ce que se sont demandés très vite les ingénieurs concevant les ordinateurs et les langages de programmation, et ils ont vite décidé de mettre en place une gestion simple de ce genre d'erreur dans ces langages.

L'idée intuitive est qu'une lettre et un âge, ce "n'est pas la même chose". On a donc introduit une sorte de classification des objets que manipule le langage : à chaque valeur du langage, on associe une étiquette qui dit "c'est une lettre", "c'est un âge" (ou "c'est un nombre"), "c'est une liste de lettres suivie par deux nombres entiers, puis un nombre à virgule", etc.

Cette étiquette s'appelle le type : le type d'une valeur est tout simplement le groupe d'objets auxquels il appartient.

Ensuite, on précise quand on déclare une variable à quel type elle appartient, et, quand on définit une opération (par exemple, la mise en minuscules), on précise (dans le code source du programme) sur quel type de données elle a le droit d'agir.

4. Le typage, une expression sémantique

La démarche du typage s'inscrit dans une direction plus générale, que l'on peut observer en de nombreux endroits de l'histoire des langages de programmation : on a apporté du sens au langage.

En introduisant le typage, on a permis au programmeur de rajouter des informations (dans le code source), pour en quelque sorte augmenter les connaissances de l'ordinateur au sujet du programme. Plus l'ordinateur est au courant, mieux il peut vérifier que ces informations sont cohérentes, et vous avertir si cela n'est pas le cas.

Le fait de modéliser le sens des objets manipulés par un programme est généralement appelé de la sémantique.

La plupart des langages informatiques se sont préoccupés de sémantique. Le C (créé pour être portable, et donc avoir des instructions reflétant mieux (qu'en assembleur) le sens des fonctions désirées, en faisant abstraction de l'architecture matérielle de l'ordinateur) et le XHTML (dont une des grandes avancées a été de mettre l'accent sur les balises décrivant la structure des pages web - et donc leur sens -, au lieu de leur présentation) en sont de bons exemples.

Exemple : transformation d'une lettre majuscule en minuscule

```
char tolower(const char c)
{
          if ( c >= 'A' && c <= 'Z' )
                return (c - 'A' + 'a');
          else
                return c;
}
```

Si un informaticien lit ce code, il comprendra immédiatement ce que fait cette fonction. Mais un des intérêts du typage repose dans le fait qu'en pratique, on n'est pas obligé de lire tout le code !

En effet, il est possible en C d'insérer avant le code d'une fonction un prototype, qui la déclare à l'avance, et qui est souvent placé dans les fichiers de header (.h) :

```
char tolower(const char);
```

Ainsi, le typage a effectivement apporté du sens dans le code : on peut se contenter de lire le prototype d'une fonction pour la comprendre.

5. Langages dynamiques, langages statiques

Il existe deux manières répandues de procéder : le typage statique, et le typage dynamique.

La différence réside dans le moment où la vérification est effectuée : au moment où les opérations sont effectuées, ou plutôt une fois pour toutes, avant de lancer le programme ?

5.1. Un langage typé dynamiquement : Python

Le principe du typage dynamique, c'est de typer au besoin, pendant l'exécution du programme : on fait ce que demande le programme, et on vérifie à chaque opération que le type correspond bien (ce qui peut se faire assez rapidement si le système des types est simple).

C'est le typage le plus utilisé par les langages de script.

Exemple en Python :

```
def essai(test):
    if test:
       return (1 + 2)
    else:
       return (1 + "essai")
```

On a défini une fonction qui prend un booléen en argument, et renvoie (1 + 2) s'il est vrai, et (1 + "essai") s'il est faux. Il y a une erreur de typage (on ne peut pas ajouter un entier et un mot !) mais l'interpréteur n'a pas protesté. On essaie ensuite d'utiliser la fonction :

```
>>> essai(True)
3
>>> essai(False)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

essai(True) renvoie un résultat correct, et seul essai(False) renvoie une erreur. Cela veut dire que tant qu'on n'est pas entré dans la branche "else" du if, aucune vérification de typage n'a été faite sur son contenu : on a véritablement un typage dynamique.

Ce typage est très simple à comprendre et à mettre en œuvre. Cependant, un de ses désavantages apparents ici est le fait que pour être sûr que les types du programme sont corrects, il faut tester tous les "embranchements" du code. Ici, il faut faire deux tests, mais pour peu que vous ayez plusieurs if et des conditions un peu compliquées, tester l'ensemble du code devient vite long et difficile (même s'il existe des outils pour tenter d'automatiser le processus).

5.2. Un langage typé statiquement : C

Un langage est typé statiquement si, avant de s'exécuter, le code source passe par une phase de vérification des types. Si le langage est compilé, cette phase est souvent comprise dans le compilateur, mais il existe aussi des langages interprétés (ou des implémentations interprétées de langages, plus exactement) qui ont un typage statique.

Un exemple de problème de typage en C est la comparaison d'un entier de type "unsigned int" (un type qui ne peut contenir que des entiers positifs) et "int" (ou "signed int"), qui peut représenter des nombres négatifs.

```
#include <stdio.h>

const unsigned int i = 23;
const int j = -5;

int main(void)
{
        if ( i < j )
             printf("i < j\n");
        else
             printf("i > j\n");

        return 0;
}
```

à la compilation (avec les warning activés) un message équivalent à celui-ci apparaît :

- `test.c: In function 'main':`
- `test.c:8: warning: comparison between signed and unsigned`

Le compilateur a repéré l'erreur de type, mais il compile le programme quand même. A l'exécution, le programme affichera que i est plus petit que j !

L'avantage du typage statique, c'est qu'il nous prévient de l'erreur avant l'exécution. De plus, il vérifie tout le code, même celui qui se trouve dans des branches de "if".

Le typage statique présente aussi un avantage non négligeable du point de vue des performances : une fois que le test de typage a été effectué, le programme est supposé valide, et le compilateur peut alors se débarrasser des informations de typage qu'il a accumulées. À l'inverse, les typages dynamiques imposent de conserver pendant toute l'exécution les informations sur le type de chaque variable, et de faire des tests tout au long du programme, ce qui a un impact sur les performances.

6. Typage fort et typage faible

Dans le monde du typage, il existe un grand nombre de manières différentes de typer, et tous les langages le font différemment, avec plus ou moins de contraintes. Ainsi, même les langages que l'on désigne couramment comme "non typés" (comme Lisp) ont généralement une certaine forme (très réduite) de typage.

L'opération de division / peut s'effectuer sur des entiers, et provoque une erreur quand le deuxième argument est nul (division par zéro). Dans un langage complètement typé (au sens mathématique du terme), il faudrait définir un type "entier non nul" qui serait celui du deuxième argument de la division, et il faudra convertir, et n'autoriser que des variables de ce type (ou faire une conversion implicite).

Le typage se mesure donc en degrés, en niveaux, et assez approximativement. Les langages les plus typés actuellement sont désignés comme pratiquant un typage fort, alors qu'on parle pour les autres (par exemple pour le langage C ou le PHP) de typage faible. Ces notions sont indépendantes de l'aspect statique ou dynamique du typage.

La différence entre le typage fort et le typage faible n'est pas très claire, car il n'y a pas de consensus à ce sujet. Une définition possible du typage faible est la suivante : les langages faiblement typés sont les langages qui n'attachent pas une grande importance au typage. Par exemple, sur une erreur de typage en PHP, dans la plupart des cas il utilisera un comportement par défaut (ignorer la variable qui pose problème, ou la convertir automatiquement vers un autre type, etc.).

7. Types énumérés

Des langages permettent au développeur de définir des types spécifiques à son application. Les types énumérés correspondent à des ensembles « finis » de valeurs possibles pour une variable. Le code suivant illustre la définition d'un nouveau type, suivie de la déclaration d'une variable typée :

```
type couleur : {rouge, bleu, vert};
var c : couleur;    // La variable c, du type couleur, peut prendre les valeurs
                    // rouge, bleu ou vert uniquement.
```

Remarques :

- Par construction, tout type de donnée informatique est de domaine fini, c'est-à-dire en bijection avec un sous-ensemble strict de l'ensemble des entiers (naturels). Par exemple, un type entier peut être défini sur 32 bits, soit une plage de valeurs entières variant de 0 à 2^{32}-1 pour un type non signé, ou une plage de -2^{31} à $+2^{31}$-1 pour un type signé.
- Le caractère « fini » d'un type énuméré (dit aussi : énumératif) s'assimile ici à un ensemble de valeurs (homogènes ou non) en bijection avec une partie stricte des entiers naturel. En pratique, le cardinal de cet ensemble est plutôt de l'ordre de la dizaine (exemple : palette de couleurs), voire du millier (exemple : plage des ports logiciels standards).