SE202 : projet de compilation Tiger

Samuel Tardieu
Année scolaire 2016/2017

Présentation du projet

Grands principes

- Tiger : langage de programmation “jouet” inventé par Andrew W. Appel pour ses livres “Modern compiler implementation in {C, Java, ML}”.
- Nous écrirons un compilateur d’un sous-ensemble de Tiger en Python ciblant l’architecture ARM.
- Certaines parties seront fournies (structures de données, squelettes), certaines ne pourront pas être modifiées.

Tiger

Tiger : le langage

- langage statiquement typé (nous nous contenterons pour l'instant du type `int` représentant des entiers sur 32 bits) ;
- permet de définir des variables, des fonctions (ce que nous implémenterons) ;
- permet de définir des types, des tableaux, des structures (ce que nous n'implémenterons pas).

Tiger : exemple de programme

Un programme Tiger sera composé pour nous d'une expression.

```
1 + 17
```

sachant qu'on peut définir des variables temporairement (de leur définition jusqu'à la fin du `end` correspondant)

```
let
  var a: int := 10  // Type explicite
  var b := 5 + a    // Type implicite
in
  a + b * 3
end
```

Tiger : exemple de `let` imbriqués

```
let
  var a := 5
in
  let
    var a := 6    /* Cache la définition précédente */
  in
    a + a         // Vaut 12, le a le plus "proche"
  end             // est utilisé
end
```

Vous aurez deviné que les commentaires sont préfixés par `//` ou sont compris entre `/*` et `*/`. Ils peuvent être imbriqués.

Tiger : exemple de fonctions

Des fonctions peuvent être définies et peuvent être récursives :

```
let
  function fact(n: int): int =
    if n < 3
    then n
    else n * fact(n-1)
in
  fact(5)        // 120
end
```

Tiger : le langage

- Les opérateurs binaires sont `+`, `-`, `*`, `/`, `<`, `<=`, `>`, `>=`, `=`, `<>` (différent), `&` (et logique), '|' (ou logique) et sont associatifs à gauche.
- Du plus prioritaire au moins prioritaire, les ordres sont (par groupe) : les parenthèses, le `-` unaire, `*` et `/`, `+` et `-` (binaire), les opérateurs de comparaison, `&`, puis `|`.
- Il existe un opérateur unaire `-`.
- Le résultat d'une comparaison entre deux valeurs renvoie soit 1 (comparaison vraie) soit 0 (comparaison fausse).
- Lors d'un test dans in `if`, toute valeur différente de 0 est vraie, 0 étant la valeur fausse.

Tiger : le langage

- Les mots clés sont `array`, `break`, `do`, `else`, `end`, `for`, `function`, `if`, `in`, `let`, `nil`, `of`, `then`, `to`, `type`, `var` et `while`.
- Les identifiants commencent par une lettre (majuscule ou minuscule) qui peut être suivie de lettres, chiffres ou *underscores* (_). Ils ne doivent pas correspondre à un mot clé. La châsse (*case*) est significative, `a` et `A` sont deux identifiants différents.
- Les nombres sont des entiers en décimal qui ne doivent pas comporter de 0 initiaux inutiles.
- La déclaration des types n'est obligatoire que dans les paramètres des déclarations de fonction, et facultatifs pour la déclaration de variable et le type de retour des déclarations de fonctions.

Tiger : le langage

Exemples :

```
let
  var abc12__34x := 0          // Autorisé
  var abc12__34x: int := 0     // Autorisé
  var _abc := 0                // Interdit
  var abc := 012               // Interdit
  var in := 0                  // Interdit
in
  ...
end
```

Python

Python : généralités

- Langage conçu à la fin des années 1980 par Guido van Rossum, premier interpréteur disponible en février 1991
- Version majeure 2.x utilisée par la majorité des projets depuis 2000.
- Version majeure 3.x en cours d'adoption (à cause de certaines incompatibilités avec Python 2.x) depuis 2008.

Nous utiliserons pour le projet Python 3.4. Cette présentation succincte n'évitera pas la nécessité de suivre un tutoriel et la pratique du langage.

Python : grands principes

Les blocs sont basés sur l'indentation :

```
def max(a, b):
  "Return the maximum of a and b"
  if a > b:
    print("a is greater")
    return a
  else:
    print("b is greater")
    return b
```

La chaîne facultative qui commmence la fonction est sa documentation (système d'aide intégré).

Python : types de base

Les types de base de Python dont nous aurons besoin sont :

- les entiers ;
- les chaînes de caractères (délimitées par `"` ou `'` au choix, ou par `"""` ou `'''` permettant de faire des chaînes multi-lignes) ;
- les listes, délimitées par des crochets, qui sont manipulables (ajout, suppression, etc.) ;
- les tuples, délimités par des parenthèses, qui ne sont pas modifiables ;
- une valeur spéciale `None` est la seule de son type, et représente traditionnellement l'absence de valeur (le `null` de Java).

Les types en Python ne sont pas explicités. Une variable peut changer de type si on lui affecte une valeur d'un autre type.

Python : exemple

```
i = 3
s = "chaîne de caractère"
s = 'chaîne avec " dedans'
s = '''chaîne de caractères
multiligne'''
l = ["ceci", "est", "une", "liste"]
```

Python : interpréteur interactif

```
% python
Python 3.4.3 (default, Sep  7 2015, 15:40:35)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more i
>>> 1+2
3
>>>
```

Python : manipulation de listes

- On peut prendre un élément d'une liste (numérotés à partir de 0), ou une sous-liste en précisant le début (inclusif) et la fin (exclusive).
- Une valeur négative pour l'index compte à partir de la fin (-1 représente le dernier élément).
- L'index de début peut être omis (*depuis le début de la liste*), celui de fin aussi (*jusqu'à la fin incluse*)

```
l = [10, 20, 30, 40, 50]
l[0]   => 10
l[-1]  => 50
l[-2]  => 40
l[2:4] = [30, 40]
l[:-1] = [10, 20, 30, 40]
l[3:] = [40, 50]
```

Python : programmation orientée objet

On peut déclarer des classes en Python. Contrairement à Java ou C++, le `this` n'est pas implicite, s'appelle (conventionnellement) `self` en python et doit être explicité. Le constructeur s'appelle `__init__`.

```
class Animal:

  def __init__(self, name):   # Constructeur
    self.name = name          # Création de champ

  def hello(self):            # Méthode
    print("Hello, I am " + self.name)

>>> a = Animal("Georges")
>>> a.hello()
Hello, I am Georges
```

Python : déclaration d'une classe fille

```
class Zebra(Animal):          # Dérive de Animal

  def __init__(self, name):
    # super() désigne l'instance courante mais
    # vu comme étant de la classe parent
    # (ici Animal)
    super().__init__(name + " the zebra")
```

Le constructeur `__init__` est surchargé et masque celui de `Animal`, et `Zebra` hérite de la méthode `hello`.

```
>>> z = Zebra("Marc")
>>> z.hello()
Hello, I am Marc the zebra
```

Python : modules

La directive `import` permet, sous plusieurs formes, d'ajouter la visibilité sur d'autres modules (prédéfinis, bibliothèques, ou parties du projet) sous plusieurs formes :

```
import mymodule
# On peut maintenant utiliser mymodule.func() si
# le module mymodule définit func()
import mymodule as foo
# On peut maintenant utiliser foo.func()
from mymodule import func
# On peut maintenant utiliser func() directement
from mymodule import *
# Importe tout de mymodule, on peut utiliser func()
```

Python : à faire

Ce que vous devez maintenant faire :

- lire le tutorial de Python 3.4 ;
- lire la documentation de virtualenvwrapper qui vous servira à créer un environnement de développement en Python 3.4 dans lequel vous pourrez installer les bibliothèques nécessaires pour le projet ;
- lire la documentation du module Ply que nous utiliserons comme lexer et parser.

Vous devez absolument maîtriser Python pour pouvoir commencer le projet.

Pour commencer le projet

Formalités

Avant de commencer le projet, vous devrez (si ce n'est pas fait) :

- demander à rejoindre à la liste de diffusion se202-2017@googlegroups.com, sur laquelle toutes les instructions seront envoyées ;
- demander à rejoindre le groupe `SE202_2017` sur le Gitlab de Télécom.

Il faudra aussi vous familiariser avec git qui sera utilisé en permanence.

Il est fortement conseillé d'utiliser git en mode `ssh` (plutôt que `https`), afin de ne pas avoir à retaper son mot de passe à chaque fois.