Exemples de programmes `Tiger`

Suzanne Collin, Sébastien Da Silva, Pierre Monnin

2018 – 2019

Introduction

Ce document a pour objectif de fournir un guide de progression dans le support des structures du langage `Tiger` lors de l'étape de *génération de code*. Les codes présentés ne sont que des exemples et ne couvrent pas forcément l'ensemble des structures du niveau associé. **Le "niveau" ne donne qu'une indication générale sur votre progression pour l'étape de *génération de code* et n'est en rien révélateur de la note finale de votre projet qui dépend également des autres évaluations (soutenances et rapports).**

L'affichage (`print`) peut être implémenté en parallèle de n'importe quel niveau. Il est préférable d'assurer son support le plus tôt possible afin de pouvoir aisément montrer des résultats en soutenance.

Niveau 1 : Programme principal, variables et structures de contrôle

- Exécution du programme principal
- Définitions de variables de types simples (`int`)
- Structures de contrôle : `if`, `while`, `for`

```
let
  var total := 0
  var i := 0
  var n := 10
  var testvar : int := 0
in
  testvar := i <= n;
  while testvar do
  (
    total := total + i * i;
    if i = 5 then
      print(5);
    testvar := i <= n
  );
  print(total)
end
```

Programme 1 – Somme des `n` premiers carrés

Niveau 2 : Appels de fonctions

— Définitions de fonctions
— Appels de fonctions et retours de valeurs

```
let
  var resultat : int := 0

  function puissance(a : int, b : int) : int =
  let
    var i := 0
    var resultat := 0
  in
    if b = 0 then
      1
    else
    (
      i := 1;
      resultat := a;
      while i < b do
      (
        resultat := resultat * a;
        i := i + 1
      );
      resultat
    )
  end

  function factorielle(n : int) =
  let
    var i := 1
  in
    if n = 0 then
      resultat := 1
    else
    (
      resultat := 1;
      while i <= n do
      (
        resultat := resultat * i;
        i = i + 1
      )
    )
  end
in
  print(puissance(3,3));
  factorielle(5);
  print(resultat)
end
```

Programme 2 – Quelques fonctions mathématiques

Niveau 3 : Récursivité et imbrication de blocs

— Fonctions récursives
— Imbrication et type de blocs

```
let
  function fibonacci(generations : int) : int =
  let
    function computation(generations : int, previous : int, pprevious : int) : int =
      if generations = 0 then
        previous
      else
        computation(generations - 1, previous + pprevious, previous)
  in
    if generations = 0 | generations = 1 then
      generations
    else
      computation(generations - 2, 1, 0)
  end

  function factorielle_recursive(n : int) : int =
  let
    function computation(n : int, acc : int) : int =
      if n = 0 then
        acc
      else
        computation(n - 1, acc * n)
  in
    computation(n, 1)
  end
in
  print(fibonacci(12));
  print(factorielle_recursive(5))
end
```

Programme 3 – Suite de Fibonacci et Factorielle récursive

Niveau 4 : Définitions de types, tableaux et structures

— Définitions d'alias
— Tableaux (`array`)
— Structures (`{ ··· }`)

```
let
  type vector = {x : int, y : int}
  type data = {v : vector, to_add : int}
  type dataArray = array of data

  var d1 := data {v = vector {x = 0 , y = 0}, to_add = 1}
  var d2 := data {v = vector {x = 1 , y = 1}, to_add = 1}
  var d3 := data {v = vector {x = 2 , y = 2}, to_add = 0}
  var d4 := data {v = vector {x = 3 , y = 4}, to_add = 1}

  var l := dataArray [4] of nil
  var result := vector {x = 0, y = 0}

  function add_vectors(l : dataArray, size : int, result : vector) =
    for i := 0 to size - 1 do
      if l[i].to_add then
      (
        result.x = result.x + l[i].v.x;
        result.y = result.y + l[i].v.y
      )
in
  l[0] := d1;
  l[1] := d2;
  l[2] := d3;
  l[3] := d4;

  add_vectors(l, 4, result);
  print(result.x);
  print(result.y)
end
```

Programme 4 – Somme de vecteurs