

Introduction au Langage

Pablo de Oliveira <pablo.oliveira@uvsq.fr>

March 27, 2015

Langage

Introduction

- ▶ Proposé par Andrew Appel en 98 dans ses livres “Modern Compiler implementation in . . .”
- ▶ C’est un langage
 - ▶ Impératif
 - ▶ Typé
 - ▶ Types de base: entiers et chaînes
 - ▶ Types dérivés: tableaux et structures
 - ▶ Fonctions imbriquées

Hello World

```
print("Hello World!")
```

Factorielle

```
let
  function fact(n : int) : int =
    if 1 < n then n * fact(n - 1) else 1
in
  print_int(fact(7))
end
```

Blocs let-in-end

```
let
  /* Declarations */
  type celsius = int
  var thermostat : celsius := 17;
  function incrementer( n : int) : int = n + 1
in
  /* Expressions */
  thermostat := incrementer(thermostat)
end
```

Fonctions

```
let
  function fonction (a1 : int, a2 : string) : int = ...
  function fonctionVoid (a1 : int, a2 : string) = ...
in
  ...
end
```


Variables

```
let
```

```
  var a : int := 0    /* OK */  
  var b := 1         /* OK */  
  var c := nil       /* NOT OK */
```

Types

- ▶ Deux types primitifs:
 - ▶ `int` entiers signés 32 bits de -2^{31} à $2^{31} - 1$.
 - ▶ attention notre parseur (pour simplifier) ne gère pas -2^{31}
 - ▶ `string` chaîne de caractères
- ▶ Des types dérivés:
 - ▶ tableaux (array)
 - ▶ structures (record)
 - ▶ alias

Tableaux

```
let
  type int_array = array of int
  var tableau := int_array[100] of 0
in
  tableau[31] := 15;
  tableau[32] := tableau[33] + tableau[34]
end
```

Structures

```
let
  type personne = {nom: string, telephone: string}
  var dupont : personne :=
      personne {nom = "dupont",
                telephone = "+33122334455"}
in
  print(dupont.telephone)
end
```

Tableaux de Structures

```
let
  type personne = {nom: string, telephone: string}
  type tannuaire = array of personne
  var annuaire: tannuaire := tannuaire[100] of nil
in
  for i := 0 to 100 do
    annuaire[i] :=
      personne {nom = get_name(),
               telephone = get_phone()}
  end
```

Alias

```
let
  type celsius = int
  type fahrenheit = int

  var c : celsius = 27
  var f : fahrenheit = 0

  function ctof(c: celsius) : fahrenheit = ...

in
  /* Attention: legal */
  f := c
end
```

Fonctions imbriquées

```
let
  type int_list = {value: int, next: int_list}
  function sum(a : int_list): int =
    let
      function reduce(a : int_list,
                      sum : int) : int =
        if a.next == nil then
          sum + a.value
        else
          reduce(a.next, sum + a.value)
        in
          reduce(a, 0)
    end
in
  ...
end
```

Fonctions récursives

```
let
  function odd(n : int) : int =
    if n == 0 then 0 else even(n-1)
  function even(n : int) : int =
    if n == 0 then 1 else odd(n-1)
in
  if odd(5) then print("5 est impair")
```


Primitives

```
print(s : string)    print_int(i : int)
```

```
getchar() : string
```

```
ord(s : string) : int
```

```
chr(i : int) : string
```

```
size(s : string) : int
```

```
concat(s1 : string, s2 : string) : string
```

```
substring(s : string, f : int, n : int) : string
```

```
not(i : int) : int
```

```
exit(code : int)
```

Exercices

- ▶ Ecrire une fonction qui donne le nieme terme de la suite de Fibonacci
- ▶ Ecrire une fonction qui trouve le plus grand élément d'un tableau
- ▶ Ecrire une fonction qui calcule la profondeur d'un arbre binaire
- ▶ Ecrire un tri-bulle sur un tableau