Le langage EINS (Eins Is Not Scala)

Types de données :

- un type de base : *Int*,
- les types fonctionnels,
- les classes.

Deux constructions de contrôle :

- *while* (<*cond*>) { ... }
- ***if*** (<*cond*>) <*expr*> [***else*** <*expr*>]

Opérateurs classiques :

- arithmétiques : $+$, $-$, $*$, $/$, $\%$,
- relationnels : $==$, $!=$, $<$, $\leq$, $>$, $\geq$,
- logiques : $!$, $\&$, $|$.

Exemples

La factorielle en impératif :

```
def fact(x: Int): Int = {
    var p: Int = 1;
    while (x > 0) {
        let p = p * x;
        let x = x - 1;
    }
    p
};

printInt(fact(5))
```

La factorielle en fonctionnel :

```
def fact(x: Int): Int = if (x == 0) 1 else x * fact(x - 1);

printInt(fact(5))
```

Les types fonctionnels

En Eins, les fonctions sont des valeurs comme les autres qui peuvent par exemple être passées à une fonction ou retournées par une fonction. Elles possèdent donc aussi un type. Leur syntaxe est donnée par la règle suivante :

$$
\begin{array}{lll}
\mathit{Type} & = & \text{"("}\ \mathit{Types}\ \text{")"}\ \text{"}\Rightarrow\text{"}\ \mathit{Type} \\
 & | & \ldots \\
\mathit{Types} & = & [\ \mathit{Type}\ \{\ \text{","}\ \mathit{Type}\ \}\ ]
\end{array}
$$

Example: Le type de la fonction

$$\textbf{\textit{def}}\ f(\mathit{arg}_1 : T_1,\ \ldots,\ \mathit{arg}_n : T_n\ ) \colon T_r = \ldots$$

est noté

$$(T_1,\ \ldots,\ T_n\ ) \Rightarrow T_r$$

Les types fonctionnels : exemples

Application de fonctions entières :

$$\textbf{def}\ apply\,(f\colon\ (Int\,) \Rightarrow Int,\ x\colon Int\,)\colon Int = f(x);$$

$$\textbf{def}\ square\,(x\colon Int\,)\colon Int = x * x;$$

$$printInt\,(apply\,(square,\ 5\,)\,)$$

Composition de fonctions :

$$\textbf{def}\ compose\,(f\colon\ (Int\,) \Rightarrow Int,\ g\colon\ (Int\,) \Rightarrow Int,\ x\colon Int\,)\colon Int = f(g\,(x\,)\,);$$

$$\textbf{def}\ square\,(x\colon Int\,)\colon Int = x * x;$$
$$\textbf{def}\ succ\,(x\colon Int\,)\colon Int = x + 1;$$

$$printInt\,(compose\ \ (square,\ succ,\ 5\,)\,)$$

Les classes

Déclarer une classe :

```
class Q {
    val num: Int;
    val den: Int;
}
```

Créer un objet :

```
new Q(2, 3)
```

Sélectionner un champ :

```
p.num
p.den
```

Définir une opération :

```
def add(p: Q, q: Q): Q =
    new Q(p.num * q.den + q.num * p.den, p.den * q.den);
```

Exemple de classe récursive : les listes

Une liste représente un ensemble ordonnée de valeurs.

```
class List {
    val head: Int;
    val tail: List;
}
```

Créations de listes :

```
null
new List(1, null)
new List(2, new List(1, null))
...
```

Accès aux valeurs d'une liste l :

```
l.head
l.tail.head
l.tail.tail.head
...
```

Fonctions sur les listes

Tester si une liste est vide :

```
def isEmpty(xs: List): Int =
  if (xs == null) true else false;
```

Longueur d'une liste :

```
def length(xs: List): Int =
  if (isEmpty(xs)) 0 else 1 + length(xs.tail);
```

Appliquer une fonction à une liste :

```
def map(f: (Int) ⇒ Int, xs: List): List =
  if (isEmpty(xs))
    null
  else
    new List(f(xs.head), map(f, xs.tail));

def square(x: Int): Int = x * x;

map(square, new List(1, new List(2, null)));
```

Les classes et fonctions prédéfinies

La classe *Unit* :

```
class Unit { }
```

Listes d'entiers :

```
class List {
    val head: Int;
    val tail: List;
}
```

Lecture et écriture d'entiers :

```
def readInt(): Int = ...
def printInt(i: Int): Unit = ...
```

Lecture et écriture de caractères :

```
def readChar(): Int = ...
def printChar(c: Int): Unit = ...
```

Syntaxe lexicale

input = { *inputelement* }
inputelement = *whitespace*
| *comment*
| *token*

token = *ident*
| *number*
| ... \{/∗ à compléter ∗/\}
comment = ”/” ”/” { *cchar* }
ident = *letter* { *letter* | *digit* | ”_” }
number = ”0” | *digit1* { *digit* }
string = ”\”” { *schar* } ”\””

whitespace = ” ” | ”\t” | ”\f” | ”\n”
letter = ”a” | ... | ”z” | ”A” | ... | ”Z”
digit = ”0” | *digit1*
digit1 = ”1” | ... | ”9”
cchar = \{*tout caractère excepté* ”\n”\}.
schar = \{*tout caractère excepté* ”\n” *et* ”\””\}.

Syntaxe (1)

$$
\begin{array}{lcl}
\textit{Program} & = & \{\ \textit{Declaration}\ \}\ \textit{Expression} \\
\textit{Declaration} & = & \textit{FunDecl} \\
 & | & \textit{ClassDecl} \\
\textit{FunDecl} & = & \text{"}\textbf{def}\text{"}\ \textit{ident}\ \text{"("}\ \textit{Formals}\ \text{")"}\ \text{":"}\ \textit{Type}\ \text{"="}\ \textit{Expression}\ \text{";"} \\
\textit{Formal} & = & \textit{ident}\ \text{":"}\ \textit{Type} \\
\textit{Formals} & = & [\ \textit{Formal}\ \{\ \text{","}\ \textit{Formal}\ \}\ ] \\
\textit{ClassDecl} & = & \text{"}\textbf{class}\text{"}\ \textit{ident}\ \text{"\{"}\ \{\ \textit{Field}\ \}\ \text{"\}"} \\
\textit{Field} & = & \text{"}\textbf{val}\text{"}\ \textit{Formal}\ \text{";"} \\
\textit{Type} & = & \text{"}\textit{Int}\text{"} \\
 & | & \textit{ident} \\
 & | & \text{"("}\ \textit{Types}\ \text{")"}\ \text{"}\Rightarrow\text{"}\ \textit{Type} \\
\textit{Types} & = & [\ \textit{Type}\ \{\ \text{","}\ \textit{Type}\ \}\ ]
\end{array}
$$

Syntaxe (2)

$$
\begin{array}{lcl}
\textit{Statement} & = & \text{"}\textit{while}\text{"}\ \text{"}(\text{"}\ \textit{Expression}\ \text{"})\text{"}\ \text{"}\{\text{"}\ \{\ \textit{Statement}\ \}\ \text{"}\}\text{"} \\
& | & \text{"}\textbf{\textit{var}}\text{"}\ \textit{Formal}\ \text{"}=\text{"}\ \textit{Expression}\ \text{";"} \\
& | & \text{"}\textbf{\textit{let}}\text{"}\ \textit{ident}\ \text{"}=\text{"}\ \textit{Expression}\ \text{";"} \\
& | & \text{"}\textbf{\textit{do}}\text{"}\ \textit{Expression}\ \text{";"} \\
\textit{Expression} & = & \text{"}\textbf{\textit{if}}\text{"}\ \text{"}(\text{"}\ \textit{Expression}\ \text{"})\text{"}\ \textit{Expression}\ [\ \text{"}\textbf{\textit{else}}\text{"}\ \textit{Expression}\ ] \\
& | & \textit{OrExpression} \\
\textit{OrExpression} & = & \textit{AndExpression} \\
& | & \textit{OrExpression}\ \text{"}|\text{"}\ \textit{AndExpression} \\
\textit{AndExpression} & = & \textit{CmpExpression} \\
& | & \textit{AndExpression}\ \text{"}\&\text{"}\ \textit{CmpExpression} \\
\textit{CmpExpression} & = & [\ \textit{SumExpression}\ \textit{CompOp}\ ]\ \textit{SumExpression} \\
\textit{CompOp} & = & \text{"}==\text{"}\ |\ \text{"}!=\text{"}\ |\ \text{"}<\text{"}\ |\ \text{"}>\text{"}\ |\ \text{"}\leq\text{"}\ |\ \text{"}\geq\text{"}
\end{array}
$$

Syntaxe (3)

$$
\begin{array}{lcl}
\mathit{SumExpression} & = & \mathit{Term} \\
 & | & \mathit{SumExpression}\ \mathit{SumOp}\ \mathit{Term} \\
\mathit{SumOp} & = & \text{"+"} \mid \text{"−"} \\
\mathit{Term} & = & [\ \mathit{Term}\ \mathit{ProdOp}\ ]\ [\ \mathit{NegateOp}\ ]\ \mathit{Factor} \\
\mathit{NegateOp} & = & \text{"−"} \mid \text{"!"} \\
\mathit{ProdOp} & = & \text{"*"} \mid \text{"/"} \mid \text{"\%"}
\end{array}
$$

Syntaxe (4)

Factor = *ident*
| *number*
| *string*
| "*true*" | "*false*"
| "**null**"
| "(" *Expression* ")"
| "{" { *Statement* } [*Expression*] "}"
| *Factor Params*
| "**new**" *ident Params*
| *Factor* "." *ident*

Params = "(" *Expressions* ")"

Expressions = [*Expression* { "," *Expression* }]