Descrizione del linguaggio SBF

1 Struttura lessicale di SBF

Parole riservate

Le parole riservate in **SBF**sono le seguenti alfanumeriche

`False`, `True`, `if`, `else`, `do`, `while`, `def`, `return`, `val`, `var` (1)

andrebbero ordinate e i seguenti simboli

_, `:`, `=`, `=>`, `<-`, `<:`, `<%`, `>:`, `#`, `@` (2)

non serviranno tutti

Identificatori

Un identificatore ⟨*Ident*⟩ è una lettera o il carattere '_' seguiti da una sequenza arbitraria di lettere, cifre e del carattere '_'.

Caratteri di ritorno a capo

Nel linguaggio **SBF** le istruzioni possono terminare con ';' oppure '\n'.

⟨*Sep*⟩ ::= `;` | ⟨*NewLine*⟩

Il carattere di ritorno a capo '\n' è sintatticamente significativo, e corrisponde a un token ⟨*NewLine*⟩ quando le seguenti condizioni sono soddisfatte:

- il token immediatamente precedente può terminare una istruzione;
- il token immediatamente successivo può iniziare una istruzione;
- il carattere appare in una regione dove il new line è abilitato.

Complicate da spiegare, sono scritte qua. `https://scala-lang.org/files/archive/spec/2.13/01-lexical-syntax.html#identifiers`

Letterali

Ci sono letterali per numeri interi, numeri in virgola mobile, singoli caratteri, booleani, stringhe. Essi seguono le convenzioni della maggior parte dei linguaggi di programmazione.

$$\begin{aligned}\langle Literal\rangle ::= &\langle Int\rangle\\ &\langle Float\rangle\\ &\langle Char\rangle\\ &\langle Boolean\rangle\\ &\langle String\rangle\end{aligned}$$

Caratteri di spaziatura e commenti

I token possono essere separati da caratteri di spaziatura o commenti.

I commenti sono di due tipi:

- i commenti di una riga sono sequenze di caratteri che iniziano con `\\` e finiscono al termine della riga;
- i commenti multi-riga sono sequenze di caratteri che iniziano con `/*` e terminano con `*/`. Possono essere annidati, ma in modo bilanciato. Ad esempio `/*commento*/*/` non è un commento valido.

Permettiamo le trailing commas? In caso va detto qua in una sottosezione che non diventano token.

2 Struttura sintattica di SBF

- Un *programma* è una sequenza di dichiarazioni separate da token di separazione $\langle Sep\rangle$.
- Una *dichiarazione* ha una delle seguenti forme:
  - *Dichiarazione di variabili e valori.*

$$\begin{aligned}\langle Decl\rangle ::= &\texttt{val}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\ \texttt{=}\ \langle Expr\rangle\\ \mid &\texttt{val}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\\ \mid &\texttt{var}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\ \texttt{=}\ \langle Expr\rangle\\ \mid &\texttt{var}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\end{aligned}$$

    dove

$$\begin{aligned}\langle TypeSpec\rangle ::= &\langle SimpleType\rangle\\ \mid &\texttt{\&}\ \langle TypeSpec\rangle\\ \mid &\texttt{Array [}\ \langle TypeSpec\rangle\ \texttt{]}\\ \langle SimpleType\rangle ::= &\texttt{bool}|\texttt{char}|\texttt{integer}|\texttt{float}|\texttt{string}\end{aligned}$$

– *Dichiarazione di funzioni/procedure.*

$$\langle Decl \rangle ::= \texttt{def}\ \langle Ident \rangle\ \langle ParamClauses \rangle\texttt{:}\ \langle TypeSpec \rangle\ \texttt{=}\ \langle Expr \rangle$$
$$\texttt{def}\ \langle Ident \rangle\ \langle ParamClauses \rangle\texttt{:}\ \langle TypeSpec \rangle\ \texttt{=}\ \langle Block \rangle$$

L'elemento $\langle ParamClauses \rangle$ è una sequenza, non vuota, di $\langle ParamClause \rangle$:

$$\langle ParamClause \rangle ::= \texttt{(}\ \langle ListOfParameters \rangle\ \texttt{)}$$

mentre $\langle ListOfParameters \rangle$ è una sequenza, che può essere vuota, di elementi separati da virgola della forma

$$\langle Parameter \rangle ::= \langle Ident \rangle\ \texttt{:}\ \langle TypeSpec \rangle$$

- Un *blocco* è una sequenza di istruzioni racchiuse fra parentesi graffe e separate da $\langle Sep \rangle$.
- Una *istruzione* ha la forma:

$$\begin{aligned}
\langle Stmt \rangle ::= {} & \texttt{if (}\ \langle Expr \rangle\ \texttt{)}\ \langle Stmt \rangle \\
| {} & \texttt{if (}\ \langle Expr \rangle\ \texttt{)}\ \langle Stmt \rangle\ \texttt{else}\ \langle Stmt \rangle \\
| {} & \texttt{if (}\ \langle Expr \rangle\ \texttt{)}\ \langle Stmt \rangle\ \langle Sep \rangle\ \texttt{else}\ \langle Stmt \rangle \\
| {} & \texttt{while (}\ \langle Expr \rangle\ \texttt{)}\ \langle Stmt \rangle \\
| {} & \texttt{do}\ \langle Stmt \rangle\ \texttt{while (}\ \langle Expr \rangle\ \texttt{)} \\
| {} & \texttt{do}\ \langle Stmt \rangle\ \langle Sep \rangle\ \texttt{while (}\ \langle Expr \rangle\ \texttt{)} \\
| {} & \texttt{return} \\
| {} & \texttt{return}\ \langle Expr \rangle \\
| {} & \langle Block \rangle \\
| {} & \langle LExpr \rangle \\
| {} & \langle LExpr \rangle\ \langle AssignmentOp \rangle\ \langle Expr \rangle \\
| {} & \langle Ident \rangle\ \langle ParamClauses \rangle
\end{aligned}$$

- Le *left expressions* permesse nel linguaggio hanno la seguente forma.

$$\begin{aligned}
\langle LExpr \rangle ::= {} & \langle Ident \rangle \\
| {} & \langle LExpr \rangle\ \texttt{(}\ \langle Expr \rangle\ \texttt{)} \\
| {} & \texttt{*}\ \langle LExpr \rangle \\
| {} & \texttt{\&}\ \langle LExpr \rangle \\
| {} & \texttt{(}\ \langle LExpr \rangle\ \texttt{)}
\end{aligned}$$

Precedenza di puntatori e array access

- Le *right expressions* permesse nel linguaggio hanno la seguente forma.

$$
\begin{aligned}
\langle Expr \rangle ::= & \langle LExpr \rangle \\
& | \langle Literal \rangle \\
& | \langle Expr \rangle \ \langle BinOp \rangle \ \langle Expr \rangle \\
& | \langle UnOp \rangle \ \langle Expr \rangle \\
& | \langle Ident \rangle \ \langle Lists \rangle \\
& | \texttt{(} \ \langle Expr \rangle \ \texttt{)}
\end{aligned}
$$

$$
\begin{aligned}
\langle BinOp \rangle &::= \texttt{||}|\texttt{\&\&}|\texttt{+}|\texttt{-}|\texttt{*}|\texttt{/}|\texttt{\%}|\texttt{\^{}}|\texttt{==}|\texttt{!=}|\texttt{<}|\texttt{<=}|\texttt{>}|\texttt{>=} \\
\langle UnOp \rangle &::= \texttt{!} \ \texttt{-}
\end{aligned}
$$