The Language Swifty

BNF-converter

May 10, 2016

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

The lexical structure of Swifty

Identifiers

Identifiers ⟨*Ident*⟩ are unquoted strings beginning with a letter, followed by any combination of letters, digits, and the characters _ ', reserved words excluded.

Literals

Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in Swifty are the following:

```
Array   array   bool
else    false   for
func    if      in
int     of      print
return  struct  true
var     while
```

The symbols used in Swifty are the following:

```
(     )     ->
=     ,     :
{     }     ||
&&    ==    !=
<     >     <=
>=    +     -
*     /     !
[     ]     .
&
```

Comments

Single-line comments begin with `//`.
Multiple-line comments are enclosed with `/*` and `*/`.

The syntactic structure of Swifty

Non-terminals are enclosed between ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

⟨*Program*⟩ ::= ⟨*ListStmt*⟩

⟨*Decl*⟩ ::= `func` ⟨*Ident*⟩ `(` ⟨*ListPDecl*⟩ `)` `->` ⟨*Type*⟩ ⟨*Stmt*⟩
| `func` ⟨*Ident*⟩ `(` ⟨*ListPDecl*⟩ `)` ⟨*Stmt*⟩
| `var` ⟨*Ident*⟩ `=` ⟨*Expr*⟩
| `struct` ⟨*Ident*⟩
| `var` ⟨*Ident*⟩ `,` ⟨*ListIdent*⟩ `=` ⟨*Expr*⟩

⟨*PDecl*⟩ ::= ⟨*Ident*⟩ `:` ⟨*Type*⟩

⟨*ListPDecl*⟩ ::= ϵ
| ⟨*PDecl*⟩
| ⟨*PDecl*⟩ `,` ⟨*ListPDecl*⟩

⟨*ListStmt*⟩ ::= ϵ
| ⟨*Stmt*⟩ ⟨*ListStmt*⟩

⟨*ListExpr*⟩ ::= ϵ
| ⟨*Expr*⟩
| ⟨*Expr*⟩ `,` ⟨*ListExpr*⟩

⟨*ListAcc*⟩ ::= ⟨*Acc*⟩
| ⟨*Acc*⟩ `,` ⟨*ListAcc*⟩

⟨*ListIdent*⟩ ::= ⟨*Ident*⟩
| ⟨*Ident*⟩ `,` ⟨*ListIdent*⟩

⟨*ListType*⟩ ::= ⟨*Type*⟩
| ⟨*Type*⟩ `,` ⟨*ListType*⟩

⟨*Block*⟩ ::= `{` ⟨*ListStmt*⟩ `}`

⟨*Acc*⟩ ::= ⟨*Ident*⟩
| ⟨*Acc*⟩ ⟨*ArraySub*⟩
| ⟨*Acc*⟩ ⟨*StructSub*⟩

⟨*Stmt*⟩ ::= ⟨*Block*⟩
| ⟨*Decl*⟩
| ⟨*Acc*⟩ = ⟨*Expr*⟩
| ⟨*Acc*⟩ `,` ⟨*ListAcc*⟩ = ⟨*Tuple*⟩
| `while (` ⟨*Expr*⟩ `)` ⟨*Stmt*⟩
| `for` ⟨*Ident*⟩ `in` ⟨*Acc*⟩ ⟨*Stmt*⟩
| `if (` ⟨*Expr*⟩ `)` ⟨*Stmt*⟩
| `if (` ⟨*Expr*⟩ `)` ⟨*Block*⟩ `else` ⟨*Stmt*⟩
| `return` ⟨*Expr*⟩
| `print` ⟨*Expr*⟩
| ⟨*FCall*⟩

⟨*FCall*⟩ ::= ⟨*Ident*⟩ `(` ⟨*ListExpr*⟩ `)`

⟨*Expr1*⟩ ::= ⟨*Expr1*⟩ || ⟨*Expr2*⟩
| ⟨*Expr2*⟩

⟨*Expr2*⟩ ::= ⟨*Expr2*⟩ `&&` ⟨*Expr3*⟩
| ⟨*Expr3*⟩

⟨*Expr3*⟩ ::= ⟨*Expr3*⟩ == ⟨*Expr4*⟩
| ⟨*Expr3*⟩ `!=` ⟨*Expr4*⟩
| ⟨*Expr4*⟩

⟨*Expr4*⟩ ::= ⟨*Expr4*⟩ < ⟨*Expr5*⟩
| ⟨*Expr4*⟩ > ⟨*Expr5*⟩
| ⟨*Expr4*⟩ <= ⟨*Expr5*⟩
| ⟨*Expr4*⟩ >= ⟨*Expr5*⟩
| ⟨*Expr5*⟩

⟨*Expr5*⟩ ::= ⟨*Expr5*⟩ + ⟨*Expr6*⟩
| ⟨*Expr5*⟩ − ⟨*Expr6*⟩
| ⟨*Expr6*⟩

⟨*Expr6*⟩ ::= ⟨*Expr6*⟩ `*` ⟨*Expr7*⟩
| ⟨*Expr6*⟩ `/` ⟨*Expr7*⟩
| ⟨*Expr7*⟩

⟨*Expr7*⟩ ::= `-` ⟨*Expr8*⟩
| `!` ⟨*Expr8*⟩
| ⟨*Expr8*⟩

⟨*Expr8*⟩ ::= ⟨*Array*⟩
| `array (` ⟨*Expr*⟩ `,` ⟨*Expr*⟩ `)`
| ⟨*Tuple*⟩
| ⟨*Acc*⟩ ⟨*ArraySub*⟩
| ⟨*Acc*⟩ ⟨*StructSub*⟩
| ⟨*FCall*⟩
| ⟨*Constant*⟩
| ⟨*Ident*⟩
| `(` ⟨*Expr*⟩ `)`

⟨*ArraySub*⟩ ::= `[` ⟨*Expr*⟩ `]`

⟨*Array*⟩ ::= `{` ⟨*ListExpr*⟩ `}`

⟨*Tuple*⟩ ::= `(` ⟨*Expr*⟩ `,` ⟨*ListExpr*⟩ `)`

⟨*StructSub*⟩ ::= `.` ⟨*Ident*⟩

⟨*Expr*⟩ ::= ⟨*Expr1*⟩

⟨*Constant*⟩ ::= `false`
| `true`
| ⟨*Integer*⟩

⟨*Type*⟩ ::= `int`
| `bool`
| `Array of` ⟨*Type*⟩
| `(` ⟨*ListType*⟩ `)`
| `&` ⟨*Type*⟩