The Language Quadr

BNF-converter

December 16, 2010

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 Quadr

Identifiers

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

Literals

String literals ⟨*String*⟩ have the form `"`x`"`, where x is any sequence of any characters except `"` unless preceded by `\`.

Integer literals ⟨*Int*⟩ are nonempty sequences of digits.

Double-precision float literals ⟨*Double*⟩ have the structure indicated by the regular expression ⟨*digit*⟩ + '.'⟨*digit*⟩ + ('*e*''-'?⟨*digit*⟩+)? i.e. two sequences of digits separated by a decimal point, optionally followed by an unsigned or negative exponent.

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 Quadr are the following:

`boolean`	`call`	`double`
`end`	`error`	`function`
`goto`	`if`	`import`
`int`	`print`	`readDouble`
`readInt`	`return`	`void`

The symbols used in Quadr are the following:

:=	.	:
.i	.d	->
$	-	{
}	+	*
/	%	~
!	<	>
==	/=	<=
>=		

Comments

Single-line comments begin with `#`.
There are no multiple-line comments in the grammar.

The syntactic structure of Quadr

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

⟨*QCode*⟩	::=	⟨*ListImport*⟩ ⟨*ListQuadr*⟩

⟨*Import*⟩	::=	`import` ⟨*Ident*⟩

⟨*ListImport*⟩	::=	ϵ
	|	⟨*Import*⟩ ⟨*ListImport*⟩

⟨*ListQuadr*⟩	::=	ϵ
	|	⟨*Quadr*⟩ ⟨*ListQuadr*⟩

```
⟨Quadr⟩       ::=  ⟨Temp⟩ := ⟨Arg⟩ ⟨Oper⟩ ⟨Arg⟩
               |   ⟨Temp⟩ := ⟨QUnop⟩ ⟨Arg⟩
               |   ⟨Temp⟩ := ⟨Arg⟩
               |   ⟨Q-Addr⟩ := ⟨Arg⟩
               |   ⟨Temp⟩ := ⟨Q-Addr⟩
               |   if ⟨Arg⟩ ⟨QComp⟩ ⟨Arg⟩ goto ⟨QSymbol⟩
               |   goto ⟨QSymbol⟩
               |   call ⟨QSymbol⟩
               |   call ⟨QSymbol⟩ . ⟨QSymbol⟩
               |   ⟨QSymbol⟩ :
               |   function ⟨QSymbol⟩ : ⟨QSignature⟩ :
               |   function end
               |   print ⟨Arg⟩
               |   print ⟨String⟩
               |   ⟨Temp⟩ := readInt
               |   ⟨Temp⟩ := readDouble
               |   return
               |   error

⟨QTyp⟩        ::=  .i
               |   .d

⟨QSignature⟩  ::=  ⟨QBasic⟩
               |   ⟨QBasic⟩ −> ⟨QSignature⟩

⟨QBasic⟩      ::=  int
               |   double
               |   void
               |   boolean

⟨Temp⟩        ::=  ⟨QSymb⟩ ⟨QTyp⟩ ⟨Integer⟩
               |   ⟨QTyp⟩ ⟨Integer⟩

⟨Arg⟩         ::=  ⟨Temp⟩
               |   ⟨QInteger⟩
               |   ⟨QDouble⟩

⟨QSymb⟩       ::=  $
               |   ⟨QSymbol⟩

⟨QInteger⟩    ::=  ⟨Integer⟩
               |   − ⟨Integer⟩

⟨QDouble⟩     ::=  ⟨Double⟩
               |   − ⟨Double⟩

⟨Q-Addr⟩      ::=  { ⟨Temp⟩ ⟨AddrOp⟩ ⟨Integer⟩ }
```

$$\begin{array}{lll}
\langle \mathit{AddrOp} \rangle & ::= & + \\
 & | & -
\end{array}$$

$$\begin{array}{lll}
\langle \mathit{Oper} \rangle & ::= & + \\
 & | & - \\
 & | & \texttt{*} \\
 & | & \texttt{/} \\
 & | & \texttt{\%}
\end{array}$$

$$\begin{array}{lll}
\langle \mathit{QUnop} \rangle & ::= & \texttt{\~{}} \\
 & | & \texttt{!}
\end{array}$$

$$\begin{array}{lll}
\langle \mathit{QComp} \rangle & ::= & < \\
 & | & > \\
 & | & == \\
 & | & \texttt{/}= \\
 & | & <= \\
 & | & >=
\end{array}$$

$$\begin{array}{lll}
\langle \mathit{QSymbol} \rangle & ::= & \langle \mathit{Ident} \rangle \\
 & | & \langle \mathit{String} \rangle
\end{array}$$