The Language Doge

BNF-converter

July 29, 2015

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 Doge

Identifiers

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

Literals

Unsigned literals are recognized by the regular expression ["123456789"]⟨*digit*⟩∗ ('u' | 'U')

Long literals are recognized by the regular expression ["123456789"]⟨*digit*⟩∗ ('l' | 'L')

UnsignedLong literals are recognized by the regular expression ["123456789"]⟨*digit*⟩∗ ('u''l' | 'U''L')

Hexadecimal literals are recognized by the regular expression '0'('x' | 'X')(⟨*digit*⟩ | ["abcdef"] | ["ABCDEF"])+

HexUnsigned literals are recognized by the regular expression '0'('x' | 'X')(⟨*digit*⟩ | ["abcdef"] | ["ABCDEF"]) + ('u' | 'U')

HexLong literals are recognized by the regular expression '0'('x' | 'X')(⟨*digit*⟩ | ["abcdef"] | ["ABCDEF"]) + ('l' | 'L')

HexUnsLong literals are recognized by the regular expression '0'('x' | 'X')(⟨*digit*⟩ | ["abcdef"] | ["ABCDEF"]) + ('u''l' | 'U''L')

Octal literals are recognized by the regular expression '0'["01234567"]∗

OctalUnsigned literals are recognized by the regular expression ‘0’[“01234567”]∗ (‘u’ | ‘U’)

OctalLong literals are recognized by the regular expression ‘0’[“01234567”]∗ (‘l’ | ‘L’)

OctalUnsLong literals are recognized by the regular expression ‘0’[“01234567”]∗ (‘u’‘l’ | ‘U’‘L’)

CDouble literals are recognized by the regular expression (⟨*digit*⟩ + ‘.’ | ‘.’⟨*digit*⟩+)((‘e’ | ‘E’)‘−’?⟨*digit*⟩+)? | ⟨*digit*⟩ + (‘e’ | ‘E’)‘−’?⟨*digit*⟩+ | ⟨*digit*⟩ + ‘.’⟨*digit*⟩ + ‘E’‘−’?⟨*digit*⟩+

CFloat literals are recognized by the regular expression (⟨*digit*⟩+‘.’⟨*digit*⟩+ | ⟨*digit*⟩ + ‘.’ | ‘.’⟨*digit*⟩+)((‘e’ | ‘E’)‘−’?⟨*digit*⟩+)?(‘f’ | ‘F’) | ⟨*digit*⟩ + (‘e’ | ‘E’)‘−’?⟨*digit*⟩ + (‘f’ | ‘F’)

CLongDouble literals are recognized by the regular expression (⟨*digit*⟩ + ‘.’⟨*digit*⟩+ | ⟨*digit*⟩ + ‘.’ | ‘.’⟨*digit*⟩+)((‘e’ | ‘E’)‘−’?⟨*digit*⟩+)?(‘l’ | ‘L’) | ⟨*digit*⟩ + (‘e’ | ‘E’)‘−’?⟨*digit*⟩ + (‘l’ | ‘L’)

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 Doge are the following:

```
Typedef_name  amaze   auto
break         case    continue
default       do      doge
else          enum    extern
for           goto    if
iz            new     not
register      sizeof  stahp
static        struct  such
switch        union   very
volatile      while   wow
```

The symbols used in Doge are the following:

{	}	::
;	such void	such char
such short	such int	such long
such float	such double	such signed
such unsigned	such typedef	such const
,	:	(
)	[	]
*	...	?
||	&&	|
^	&	==
!=	<	>
<=	>=	<<
>>	+	−
/	%	++
−−	.	−>
~	*=	/=
%=	+=	−=
<<=	>>=	&=
^=	|=	

Comments

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

The syntactic structure of Doge

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

⟨*External-declaration*⟩ ::= `doge` ⟨*ClassName*⟩ ⟨*Extends*⟩ { ⟨*ListExternal-declaration*⟩ }
| `amaze` ⟨*Ident*⟩ { ⟨*ListExternal-declaration*⟩ }
| ⟨*Function-def*⟩
| ⟨*Dec*⟩

⟨*ClassName*⟩ ::= ⟨*Ident*⟩ `::` ⟨*Ident*⟩
| ⟨*Ident*⟩

⟨*Extends*⟩ ::= `very` ⟨*ClassName*⟩
| ϵ

⟨*Jump-stm*⟩ ::= `wow ;`
| `wow` ⟨*Exp*⟩ `;`
| `goto` ⟨*Ident*⟩ `;`
| `continue ;`
| `break ;`

⟨*Type-specifier*⟩ ::= `such void`
| `such char`
| `such short`
| `such int`
| `such long`
| `such float`
| `such double`
| `such signed`
| `such unsigned`
| ⟨*Struct-or-union-spec*⟩
| ⟨*Enum-specifier*⟩
| `Typedef_name`

⟨*Storage-class-specifier*⟩ ::= `such typedef`
| `extern`
| `static`
| `auto`
| `register`

⟨*Type-qualifier*⟩ ::= `such const`
| `volatile`

⟨*Unary-operator*⟩ ::= `not`
| `&`
| `*`
| +
| −
| `~`

⟨*Assignment-op*⟩ ::= `iz`
| `*=`
| `/=`
| `%=`
| +=
| −=
| <<=
| >>=
| `&=`
| `^=`
| |=

⟨*Init-declarator*⟩ ::= ⟨*Declarator*⟩ `iz` ⟨*Initializer*⟩
| ⟨*Declarator*⟩

⟨*Enumerator*⟩ ::= ⟨*Ident*⟩ `iz` ⟨*Constant-expression*⟩
| ⟨*Ident*⟩

⟨*Exp2*⟩ ::= ⟨*Exp15*⟩ ⟨*Assignment-op*⟩ `new` ⟨*ClassName*⟩
| `stahp` ⟨*Ident*⟩
| ⟨*Exp15*⟩ ⟨*Assignment-op*⟩ ⟨*Exp2*⟩
| ⟨*Exp3*⟩

⟨*Declaration-specifier*⟩ ::= `such` ⟨*ClassName*⟩ ⟨*Pointer*⟩ ⟨*Ident*⟩
| `such` ⟨*ClassName*⟩ ⟨*Ident*⟩
| ⟨*Type-specifier*⟩
| ⟨*Storage-class-specifier*⟩
| ⟨*Type-qualifier*⟩

⟨*Program*⟩ ::= ⟨*ListExternal-declaration*⟩

⟨*ListExternal-declaration*⟩ ::= ⟨*External-declaration*⟩
| ⟨*External-declaration*⟩ ⟨*ListExternal-declaration*⟩

⟨*Function-def*⟩ ::= ⟨*ListDeclaration-specifier*⟩ ⟨*Declarator*⟩ ⟨*ListDec*⟩ ⟨*Compound-stm*⟩
| ⟨*ListDeclaration-specifier*⟩ ⟨*Declarator*⟩ ⟨*Compound-stm*⟩
| ⟨*Declarator*⟩ ⟨*ListDec*⟩ ⟨*Compound-stm*⟩
| ⟨*Declarator*⟩ ⟨*Compound-stm*⟩

⟨*Dec*⟩ ::= ⟨*ListDeclaration-specifier*⟩ `;`
| ⟨*ListDeclaration-specifier*⟩ ⟨*ListInit-declarator*⟩ `;`

⟨*ListDec*⟩ ::= ⟨*Dec*⟩
| ⟨*Dec*⟩ ⟨*ListDec*⟩

⟨*ListDeclaration-specifier*⟩ ::= ⟨*Declaration-specifier*⟩
| ⟨*Declaration-specifier*⟩ ⟨*ListDeclaration-specifier*⟩

⟨*ListInit-declarator*⟩ ::= ⟨*Init-declarator*⟩
| ⟨*Init-declarator*⟩ `,` ⟨*ListInit-declarator*⟩

⟨*Struct-or-union-spec*⟩ ::= ⟨*Struct-or-union*⟩ ⟨*Ident*⟩ { ⟨*ListStruct-dec*⟩ }
| ⟨*Struct-or-union*⟩ { ⟨*ListStruct-dec*⟩ }
| ⟨*Struct-or-union*⟩ ⟨*Ident*⟩

⟨*Struct-or-union*⟩ ::= `struct`
| `union`

⟨*ListStruct-dec*⟩ ::= ⟨*Struct-dec*⟩
| ⟨*Struct-dec*⟩ ⟨*ListStruct-dec*⟩

⟨*Struct-dec*⟩ ::= ⟨*ListSpec-qual*⟩ ⟨*ListStruct-declarator*⟩ ;

⟨*ListSpec-qual*⟩ ::= ⟨*Spec-qual*⟩
| ⟨*Spec-qual*⟩ ⟨*ListSpec-qual*⟩

⟨*Spec-qual*⟩ ::= ⟨*Type-specifier*⟩
| ⟨*Type-qualifier*⟩

⟨*ListStruct-declarator*⟩ ::= ⟨*Struct-declarator*⟩
| ⟨*Struct-declarator*⟩ , ⟨*ListStruct-declarator*⟩

⟨*Struct-declarator*⟩ ::= ⟨*Declarator*⟩
| : ⟨*Constant-expression*⟩
| ⟨*Declarator*⟩ : ⟨*Constant-expression*⟩

⟨*Enum-specifier*⟩ ::= `enum` { ⟨*ListEnumerator*⟩ }
| `enum` ⟨*Ident*⟩ { ⟨*ListEnumerator*⟩ }
| `enum` ⟨*Ident*⟩

⟨*ListEnumerator*⟩ ::= ⟨*Enumerator*⟩
| ⟨*Enumerator*⟩ , ⟨*ListEnumerator*⟩

⟨*Declarator*⟩ ::= ⟨*Pointer*⟩ ⟨*Direct-declarator*⟩
| ⟨*Direct-declarator*⟩

⟨*Direct-declarator*⟩ ::= ⟨*Ident*⟩
| `(` ⟨*Declarator*⟩ `)`
| ⟨*Direct-declarator*⟩ `[` ⟨*Constant-expression*⟩ `]`
| ⟨*Direct-declarator*⟩ `[ ]`
| ⟨*Direct-declarator*⟩ `(` ⟨*Parameter-type*⟩ `)`
| ⟨*Direct-declarator*⟩ `(` ⟨*ListIdent*⟩ `)`
| ⟨*Direct-declarator*⟩ `( )`

⟨*Pointer*⟩ ::= *
| * ⟨*ListType-qualifier*⟩
| * ⟨*Pointer*⟩
| * ⟨*ListType-qualifier*⟩ ⟨*Pointer*⟩

⟨*ListType-qualifier*⟩ ::= ⟨*Type-qualifier*⟩
| ⟨*Type-qualifier*⟩ ⟨*ListType-qualifier*⟩

⟨*Parameter-type*⟩ ::= ⟨*Parameter-declarations*⟩
| ⟨*Parameter-declarations*⟩ , ...

⟨*Parameter-declarations*⟩ ::= ⟨*Parameter-declaration*⟩
| ⟨*Parameter-declarations*⟩ , ⟨*Parameter-declaration*⟩

⟨*Parameter-declaration*⟩ ::= ⟨*ListDeclaration-specifier*⟩
| ⟨*ListDeclaration-specifier*⟩ ⟨*Declarator*⟩
| ⟨*ListDeclaration-specifier*⟩ ⟨*Abstract-declarator*⟩

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

⟨*Initializer*⟩ ::= ⟨*Exp2*⟩
| `{` ⟨*Initializers*⟩ `}`
| `{` ⟨*Initializers*⟩ `,` `}`

⟨*Initializers*⟩ ::= ⟨*Initializer*⟩
| ⟨*Initializers*⟩ `,` ⟨*Initializer*⟩

⟨*Type-name*⟩ ::= ⟨*ListSpec-qual*⟩
| ⟨*ListSpec-qual*⟩ ⟨*Abstract-declarator*⟩

⟨*Abstract-declarator*⟩ ::= ⟨*Pointer*⟩
| ⟨*Dir-abs-dec*⟩
| ⟨*Pointer*⟩ ⟨*Dir-abs-dec*⟩

⟨*Dir-abs-dec*⟩ ::= `(` ⟨*Abstract-declarator*⟩ `)`
| `[` `]`
| `[` ⟨*Constant-expression*⟩ `]`
| ⟨*Dir-abs-dec*⟩ `[` `]`
| ⟨*Dir-abs-dec*⟩ `[` ⟨*Constant-expression*⟩ `]`
| `(` `)`
| `(` ⟨*Parameter-type*⟩ `)`
| ⟨*Dir-abs-dec*⟩ `(` `)`
| ⟨*Dir-abs-dec*⟩ `(` ⟨*Parameter-type*⟩ `)`

⟨*Stm*⟩ ::= ⟨*Labeled-stm*⟩
| ⟨*Compound-stm*⟩
| ⟨*Expression-stm*⟩
| ⟨*Selection-stm*⟩
| ⟨*Iter-stm*⟩
| ⟨*Jump-stm*⟩

⟨*Labeled-stm*⟩ ::= ⟨*Ident*⟩ `:` ⟨*Stm*⟩
| `case` ⟨*Constant-expression*⟩ `:` ⟨*Stm*⟩
| `default` `:` ⟨*Stm*⟩

⟨*Compound-stm*⟩ ::= `{` `}`
| `{` ⟨*ListStm*⟩ `}`
| `{` ⟨*ListDec*⟩ `}`
| `{` ⟨*ListDec*⟩ ⟨*ListStm*⟩ `}`

⟨*Expression-stm*⟩ ::= `;`
| ⟨*Exp*⟩ `;`

⟨*Selection-stm*⟩ ::= `if` `(` ⟨*Exp*⟩ `)` ⟨*Stm*⟩
| `if` `(` ⟨*Exp*⟩ `)` ⟨*Stm*⟩ `else` ⟨*Stm*⟩
| `switch` `(` ⟨*Exp*⟩ `)` ⟨*Stm*⟩

⟨*Iter-stm*⟩ ::= `while (` ⟨*Exp*⟩ `)` ⟨*Stm*⟩
| `do` ⟨*Stm*⟩ `while (` ⟨*Exp*⟩ `) ;`
| `for (` ⟨*Expression-stm*⟩ ⟨*Expression-stm*⟩ `)` ⟨*Stm*⟩
| `for (` ⟨*Expression-stm*⟩ ⟨*Expression-stm*⟩ ⟨*Exp*⟩ `)` ⟨*Stm*⟩

⟨*ListStm*⟩ ::= ⟨*Stm*⟩
| ⟨*Stm*⟩ ⟨*ListStm*⟩

⟨*Exp*⟩ ::= ⟨*Exp*⟩ `,` ⟨*Exp2*⟩
| ⟨*Exp2*⟩

⟨*Exp3*⟩ ::= ⟨*Exp4*⟩ `?` ⟨*Exp*⟩ `:` ⟨*Exp3*⟩
| ⟨*Exp4*⟩

⟨*Exp4*⟩ ::= ⟨*Exp4*⟩ || ⟨*Exp5*⟩
| ⟨*Exp5*⟩

⟨*Exp5*⟩ ::= ⟨*Exp5*⟩ `&&` ⟨*Exp6*⟩
| ⟨*Exp6*⟩

⟨*Exp6*⟩ ::= ⟨*Exp6*⟩ | ⟨*Exp7*⟩
| ⟨*Exp7*⟩

⟨*Exp7*⟩ ::= ⟨*Exp7*⟩ `^` ⟨*Exp8*⟩
| ⟨*Exp8*⟩

⟨*Exp8*⟩ ::= ⟨*Exp8*⟩ `&` ⟨*Exp9*⟩
| ⟨*Exp9*⟩

⟨*Exp9*⟩ ::= ⟨*Exp9*⟩ == ⟨*Exp10*⟩
| ⟨*Exp9*⟩ `!=` ⟨*Exp10*⟩
| ⟨*Exp10*⟩

⟨*Exp10*⟩ ::= ⟨*Exp10*⟩ < ⟨*Exp11*⟩
| ⟨*Exp10*⟩ > ⟨*Exp11*⟩
| ⟨*Exp10*⟩ <= ⟨*Exp11*⟩
| ⟨*Exp10*⟩ >= ⟨*Exp11*⟩
| ⟨*Exp11*⟩

⟨*Exp11*⟩ ::= ⟨*Exp11*⟩ << ⟨*Exp12*⟩
| ⟨*Exp11*⟩ >> ⟨*Exp12*⟩
| ⟨*Exp12*⟩

⟨*Exp12*⟩ ::= ⟨*Exp12*⟩ + ⟨*Exp13*⟩
| ⟨*Exp12*⟩ − ⟨*Exp13*⟩
| ⟨*Exp13*⟩

⟨*Exp13*⟩ ::= ⟨*Exp13*⟩ `*` ⟨*Exp14*⟩
| ⟨*Exp13*⟩ `/` ⟨*Exp14*⟩
| ⟨*Exp13*⟩ `%` ⟨*Exp14*⟩
| ⟨*Exp14*⟩

⟨*Exp14*⟩ ::= `(` ⟨*Type-name*⟩ `)` ⟨*Exp14*⟩
| ⟨*Exp15*⟩

⟨*Exp15*⟩ ::= ++ ⟨*Exp15*⟩
| −− ⟨*Exp15*⟩
| ⟨*Unary-operator*⟩ ⟨*Exp14*⟩
| `sizeof` ⟨*Exp15*⟩
| `sizeof (` ⟨*Type-name*⟩ `)`
| ⟨*Exp16*⟩

⟨*Exp16*⟩ ::= ⟨*Exp16*⟩ `[` ⟨*Exp*⟩ `]`
| ⟨*Exp16*⟩ `( )`
| ⟨*Exp16*⟩ `(` ⟨*ListExp2*⟩ `)`
| ⟨*Exp16*⟩ `.` ⟨*Ident*⟩
| ⟨*Exp16*⟩ −> ⟨*Ident*⟩
| ⟨*Exp16*⟩ ++
| ⟨*Exp16*⟩ −−
| ⟨*Exp17*⟩

⟨*Exp17*⟩ ::= ⟨*Ident*⟩
| ⟨*Constant*⟩
| ⟨*String*⟩
| `(` ⟨*Exp*⟩ `)`

⟨*Constant*⟩ ::= ⟨*Double*⟩
| ⟨*Char*⟩
| ⟨*Unsigned*⟩
| ⟨*Long*⟩
| ⟨*UnsignedLong*⟩
| ⟨*Hexadecimal*⟩
| ⟨*HexUnsigned*⟩
| ⟨*HexLong*⟩
| ⟨*HexUnsLong*⟩
| ⟨*Octal*⟩
| ⟨*OctalUnsigned*⟩
| ⟨*OctalLong*⟩
| ⟨*OctalUnsLong*⟩
| ⟨*CDouble*⟩
| ⟨*CFloat*⟩
| ⟨*CLongDouble*⟩
| ⟨*Integer*⟩

⟨*Constant-expression*⟩ ::= ⟨*Exp3*⟩

⟨*ListExp2*⟩ ::= ⟨*Exp2*⟩
| ⟨*Exp2*⟩ `,` ⟨*ListExp2*⟩