# The Language MUDA

BNF-converter

October 14, 2007

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 MUDA

### Identifiers

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

### Literals

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.

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

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

### 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 MUDA are the following:

```
always_inline  else    force_inline
if             in      inline
inout          new     out
return         static  struct
while
```

The symbols used in MUDA are the following:

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

### Comments

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

## The syntactic structure of MUDA

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

⟨*Prog*⟩ ::= ⟨*ListStruc*⟩ ⟨*ListFunc*⟩

⟨*ListStruc*⟩ ::= $\epsilon$
| ⟨*Struc*⟩ ⟨*ListStruc*⟩

⟨*Struc*⟩ ::= `struct` { ⟨*ListMDec*⟩ } ⟨*Ident*⟩ ;

⟨*ListMDec*⟩ ::= ⟨*MDec*⟩
| ⟨*MDec*⟩ ⟨*ListMDec*⟩

⟨*MDec*⟩ ::= ⟨*Typ*⟩ ⟨*Ident*⟩ ;

⟨*ListFunc*⟩ ::= $\epsilon$
| ⟨*Func*⟩ ⟨*ListFunc*⟩

⟨*Func*⟩ ::= ⟨*ListFuncSpec*⟩ ⟨*Typ*⟩ ⟨*Ident*⟩ `(` ⟨*ListFormalDec*⟩ `)` `{` ⟨*ListStm*⟩ `}`

⟨*ListFuncSpec*⟩ ::= $\epsilon$
| ⟨*FuncSpec*⟩ ⟨*ListFuncSpec*⟩

⟨*FuncSpec*⟩ ::= `inline`
| `force_inline`
| `always_inline`
| `static`

⟨*FormalDec*⟩ ::= ⟨*ListQual*⟩ ⟨*Typ*⟩ ⟨*Ident*⟩

⟨*ListFormalDec*⟩ ::= $\epsilon$
| ⟨*FormalDec*⟩
| ⟨*FormalDec*⟩ `,` ⟨*ListFormalDec*⟩

⟨*ListDecInit*⟩ ::= ⟨*DecInit*⟩
| ⟨*DecInit*⟩ `,` ⟨*ListDecInit*⟩

⟨*DecInit*⟩ ::= ⟨*Ident*⟩ ⟨*ListDecInitExp*⟩

⟨*ListDecInitExp*⟩ ::= $\epsilon$
| ⟨*DecInitExp*⟩

⟨*DecInitExp*⟩ ::= `=` ⟨*Exp*⟩

⟨*Qual*⟩ ::= `in`
| `out`
| `inout`

⟨*ListQual*⟩ ::= $\epsilon$
| ⟨*Qual*⟩ ⟨*ListQual*⟩

⟨*Stm*⟩ ::= ⟨*Typ*⟩ ⟨*ListDecInit*⟩ `;`
| ⟨*Ident*⟩ `=` ⟨*Exp*⟩ `;`
| ⟨*Ident*⟩ ⟨*ListField*⟩ `=` ⟨*Exp*⟩ `;`
| ⟨*Exp*⟩ `;`
| `{` ⟨*ListStm*⟩ `}`
| `while` `(` ⟨*Exp*⟩ `)` ⟨*Stm*⟩
| `if` `(` ⟨*Exp*⟩ `)` `{` ⟨*ListStm*⟩ `}` `else` `{` ⟨*ListStm*⟩ `}`
| `return` ⟨*Exp*⟩ `;`
| ⟨*Ident*⟩ `=` `new` ⟨*Integer*⟩ `;`
| ⟨*Stm*⟩ `;`

⟨*ListStm*⟩ ::= $\epsilon$
| ⟨*Stm*⟩ ⟨*ListStm*⟩

⟨*Exp*⟩ ::= ⟨*Exp*⟩ `&` ⟨*Exp1*⟩
| ⟨*Exp*⟩ `|` ⟨*Exp1*⟩
| ⟨*Exp1*⟩

⟨*Exp1*⟩ ::= ⟨*Exp1*⟩ `==` ⟨*Exp2*⟩
| ⟨*Exp1*⟩ `!=` ⟨*Exp2*⟩
| ⟨*Exp2*⟩

⟨*Exp2*⟩ ::= ⟨*Exp2*⟩ `>` ⟨*Exp3*⟩
| ⟨*Exp2*⟩ `>=` ⟨*Exp3*⟩
| ⟨*Exp2*⟩ `<` ⟨*Exp3*⟩
| ⟨*Exp2*⟩ `<=` ⟨*Exp3*⟩
| ⟨*Exp3*⟩

⟨*Exp3*⟩ ::= ⟨*Exp3*⟩ `+` ⟨*Exp4*⟩
| ⟨*Exp3*⟩ `-` ⟨*Exp4*⟩
| ⟨*Exp4*⟩

⟨*Exp4*⟩ ::= ⟨*Exp4*⟩ `*` ⟨*Exp5*⟩
| ⟨*Exp4*⟩ `/` ⟨*Exp5*⟩
| ⟨*Exp4*⟩ `/.` ⟨*Exp5*⟩
| ⟨*Exp5*⟩

⟨*Exp5*⟩ ::= `-` ⟨*Exp6*⟩
| ⟨*Exp6*⟩

⟨*Exp6*⟩ ::= ⟨*Ident*⟩
| ⟨*Ident*⟩ `.` ⟨*ListField*⟩
| ⟨*Integer*⟩
| ⟨*Double*⟩
| ⟨*CFloat*⟩
| ⟨*Hexadecimal*⟩
| ⟨*Ident*⟩ `(` ⟨*ListFuncArgs*⟩ `)`
| `(` ⟨*Exp*⟩ `)`

⟨*ListFuncArgs*⟩ ::= $\epsilon$
| ⟨*FuncArgs*⟩
| ⟨*FuncArgs*⟩ `,` ⟨*ListFuncArgs*⟩

⟨*FuncArgs*⟩ ::= ⟨*Exp1*⟩

⟨*ListField*⟩ ::= $\epsilon$
| ⟨*Field*⟩ ⟨*ListField*⟩

⟨*Field*⟩ ::= `.` ⟨*Ident*⟩

$\langle Typ \rangle \quad ::= \quad \langle Ident \rangle$