用BNF描述MUA语言

- 姓名：尹嘉权
- 学号：3120000419
- 班级：计科1205

---

lexeme 语素

- 0,1,2,3,4,5,6,7,8,9,.
- a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z
- [,],-,"
- _number : 以上述[0~9]数字任意组合而成，同时前面允许带'-'，不区分整数，浮点数
- _word : 以双引号"开头，以上述[a~z]字母任意组合而成，不含空格，采用Unicode编码
- _list : 以方括号[]包含，其中的元素以逗号,分隔；元素可是任意类型；元素类型可不一致
- _bool : 由true和false构成
- //,:,make,thing,erase,isname,print,read,readlinst,add,sub,mul,div,mod,eq,gt,lt,and,or,not,random,sqrt,isnumber,isword,islist,isbool,test,iftrue,iffalse,

---

token 标记

- 由lexeme中所有合法的 _number 构成的集合：数字 number
- 由lexeme中所有合法的 _word 构成的集合：单词 word
- 由lexeme中26个字母构成的集合：letter
- 由lexeme中所有合法的 _list 构成的集合：列表 list
- 由lexeme中_bool 构成的集合：布尔 bool
- 由lexeme中的 //,:,make,thing,erase,isname,print,read,readlinst,add,sub,mul,div,mod,eq,gt,lt,and,or,not,random,sqrt,isnumber,isword,islist,isbool,test,iftrue,iffalse, 构成的集合：operator

---

terminals 终结符

- 所有的lexeme 和 token 均为终结符

---

non-terminals 非终结符

- `<program>`
- `<set_of_stmts>` `<stmts>` `<stmt>` `<//>` `<charset>`
- `<word>` `<number>` `<bool>` `<list>` `<value>`
- `<op1_word>` `<op1_number>` `<op1_value>` `<one_word_operator>` `<one_number_operator>` `<one_value_operator>` `<number_operator>` `<numword_operator>` `<andor_operator>`
- `<op_number>` `<op_numword>` `<op_and_or>` `<op_not>`
- `<make>` `<iftrue>` `<iffalse>` `<word_link>` `<list_link>` `<first_wl>` `<last_wl>` `<butfirst_wl>` `<butlast_wl>` `<item_n_wl>`
- `<functionName>` `<arglist>` `<funciton_define>` `<function_use>`
- `<pi>`

---

start 起始符号

- `<program>`

---

production rules 生成规则

- `<program>` -> `<set_of_stmts>`
- `<set_of_stmts>` -> `<stmts>` | `<stmts>'\n'<stmts>`
- `<stmts>` -> `<stmt>` | `<stmt> <//>` | `<stmt><//>`
- `<//>` -> // `<charset>` | // `<charset>`
- `<charset>` -> `word` | `number` | `<charset> <charset>`
- `<stmt>` -> `print <value>` | `erase <word>` | `<make>` | `<iftrue>` | `<iffalse>` | `join <list> <value>` | `repeat <number> <list>` | `stop` | `wait <number>` | `save <word>` | `load <word>` | `erall` | `poall` | `<function_define>` | `<function_use>` | `output <value>` | `local <value>` | `run <list>` | `if <bool> <list> <list>`

- `<word>` -> 除operator外的所有word（这里面operator如上述token描述）

- `<number>` -> `number` | `<pi>`
- `<bool>` -> `bool`
- `<list>` -> `list`
- `<value>` -> `read` | `readlinst` | `<op1_word>` | `<op1_number>` | `<op1_value>` | `<op_number>` | `<op_numword>` | `<op_and_or>` | `<op_not>` | `<word_link>` | `<list_link>` | `<first_wl>` | `<last_wl>` | `<butfirst_wl>` | `<buflast_wl>` | `<item_n_wl>`

- `<op1_word>` -> `<one_word_operator> <word>`
- `<op1_number>` -> `<one_number_operator> <number>`
- `<op1_value>` -> `<one_value_operator> <value>`
- `<one_word_operator>` -> `thing` | `:` | `isname`
- `<one_number_operator>` -> `random` | `sqrt`
- `<one_value_operator>` -> `isnumber` | `isword` | `islist` | `isbool` | `test`
- `<number_operator>` -> `add` | `sub` | `mul` | `div` | `mod`
- `<numword_operator>` -> `eq` | `gt` | `lt`
- `<andor_operator>` -> `and` | `or`
- `<op_number>` -> `<number_operator> <number> <number>`
- `<op_numword>` -> `<numword_operator> <number> <number>` | `<numword_operator> <word> <word>`
- `<op_and_or>` -> `<andor_operator> <bool> <bool>`
- `<op_not>` -> `not <bool>`

- `<make>` -> `make <word> <value>`
- `<iftrue>` -> `iftrue <list>`
- `<iffalse>` -> `iffalse <list>`
- `<word_link>` -> `word <word> <word>` | `word <word> <number>` | `word <word> <bool>`
- `<list_link>` -> `list <list> <list>`
- `<first_wl>` -> `first <word>` | `first <list>`
- `<last_wl>` -> `last <word>` | `last <list>`
- `<butfirst_wl>` -> `butfirst <word>` | `butfirst <list>`
- `<butlast_wl>` -> `butlast <word>` | `butlast <list>`
- `<item_n_wl>` -> `item <number> <word>` | `item <number> <list>`

- `<functionName>` -> `letter<functionName>` | `letter`
- `<arglist>` -> `<list>`
- `<funciton_define>` -> `make <functionName> [<arglist> <list>]`
- `<function_use>` -> `<functionName> <arglist>`

- `<pi>` -> 3.14159