// short description of syntax:

// | - or

// ( ) - grouping

// [ ] – option (0 or 1)

// { } - repetition (0 or more)

// . - any symbol

// ‘ ‘ - terminal

// < > - ranges

program = { (non-func\_declaration ‘;’) | function\_declaration } main

main = ‘int’ ‘main’ ‘(‘ ‘)’ compound\_operator

type = ‘int’ | ‘float’ | ‘string’ | ‘bool’ | ‘char’

int = num

float = num ‘.’ [num] [ ‘E’ [‘+’ | ‘-’] num ]

bool = ‘true’ | ‘false’

char = .

string = { char }

non-func\_declaration = variable\_declaration | array\_declaration

variable\_declaration = type section {‘,’ section}

section = name [ ‘=‘ expression]

array\_declaration = array\_template name { ‘[‘ (num | variable) ‘]’ } [ ‘=’ initializer\_list ] ‘;’

array\_template = [‘dynamic’] ‘array’ ‘<’ ( type | array\_template ) ‘>’

initializer\_list = ‘{‘ [ expression { ‘,’ expression } ] ‘}’

function\_declaration = (type | ‘void’) name ‘(‘ formal\_parameter {‘,’ formal\_parameter}

{‘,’ formal\_parameter ‘=‘ constant\_exp} ‘)’ compound\_operator

formal\_parameter = type name

derived\_operator = loop\_operator | conditional\_operator

loop\_operator = while | for | do\_while

conditional\_operator = if | switch

compound\_operator = ‘{‘ { ( operator ‘;’ ) | derived\_operator | (non-func\_declaration ‘;’ ) | nothing } ‘}’

operator = return | expression | break | continue

break = ‘break’

continue = ‘continue’

return = ‘return’ [variable]

nothing = eps

digit = <0-9>

letter = <A-Z | a-z | \_>

num = digit { digit }

constant\_expr = operand\_num | operand\_bool | operand\_char | operand\_string

operand\_num = [‘+’ | ‘-’] (int | float)

operand\_bool = [!] bool

operand\_char = ‘\‘‘ char ‘\‘‘

operand\_string = ‘"‘ string ‘"‘

variable = name { ‘[’ index ’]’ }

func\_call = name ‘(‘ [non-formal\_parameter] {‘,’ non-formal\_parameter} ‘)’

non-formal\_parameter = func\_call | expression

if = ‘if’ ‘(‘ expression ‘)’ ( (operator ‘;’) | compound\_operator ) [ ‘else’ ( if | (operator ‘;’) | compound\_operator ) ]

while = ‘while’ ‘(‘ expression ‘)’ ( (operator ‘;’) | compound\_operator )

for = ‘for’ ‘(‘ [ non-func\_declaration | expression ] ‘;’ [ expression ] ‘;’ [ expression ] ‘)’

( (operator,‘;’) | compound\_operator )

do\_while = ‘do’ ( (operator ‘;’) | compound\_operator) ‘while’ ‘(‘ expression ’)’ ‘;’

switch = ‘switch’ ‘(‘ variable ‘)’ ‘{‘

{ ‘case’ constant\_expression) ‘:’ ( (operator ‘;’) | compound\_operator ) ‘;’ }

[ ‘default’ ‘:’ ( (operator ‘;’) | compound\_operator ) ] ‘}’

value = variable | constant\_expr | func\_call

expression = priority\_14

assignment = ‘=‘ | ‘+=’ | ‘-=‘ | ‘\*=’ | ‘/=’ | ‘\*\*=’ | ‘>>=’ | ‘<<=’ | ‘%=’ | ‘&=’ | ‘|=’ | ‘^=’

priority\_1 = ( ‘(‘ priority\_1 ‘)’ ) | value

priority\_2 = ( (‘++’ | ‘--') priority\_2 ) | priority\_1

priority\_3= priority\_2 [ ‘\*\*’ priority\_3 ]

priority\_4 = ( (‘!’ | ‘~’ | ‘-’ | ‘+’) priority\_4 ) | priority\_3

priority\_5 = priority\_4 [ (‘\*’ | ‘/’ | ‘%’ ) priority\_5 ]

priority\_6 = priority\_5 [ (‘+’ | ‘-’) priority\_6 ]

priority\_7 = priority\_6 [ (‘<<’ | ‘>>’) priority\_7 ]

priority\_8 = priority\_7 [ (‘<’ | ‘>’ | ‘>=’ | ‘<=’ | ‘==’ | ‘!=’) priority\_8 ]

priority\_9 = priority\_8 [ (‘==’ | ‘!=’) priority\_9 ]

priority\_10 = priority\_9 [ ‘&’ priority\_10 ]

priority\_11 = priority\_10 [ ‘^’ priority\_11 ]

priority\_12 = priority\_11 [ ‘|’ priority\_12 ]

priority\_13 = priority\_12 [ ‘&&’ priority\_13 ]

priority\_14 = priority\_13 [ ‘||’ priority\_14 ]

priority\_15 = priority\_14 [ assignment priority\_15 ]