**Tiny-handmade**

1. **The Lexer**
2. **Context-free grammar**
   1. **Rules**

<*program*> -> <*declaration*>+

<*declaration*> -> <*fun\_declaration*>

<declaration> -> <*var\_declaration*>

<*fun\_declaration*> -> <*type*> **NAME** **LPAR** <*formal\_pars*> **RPAR** <*block*>

<*formal\_pars*> -> <*formal\_par*> {**COMMA** <*formal\_par*>}\*

<*formal\_pars*> -> **ε**

<*formal\_par*> -> <*type*> **NAME**

<*block*> -> **LBRACE** <*var\_declarations*>\* <*statements*> **RBRACE**

<*var\_declarations*> -> <*var\_declarations*>\* <*var\_declaration*>

<*var\_declaration*> -> <*type*> **NAME** **SEMICOLON**

<*type*> -> **INT**

<*type*> -> **CHAR**

<*type*> -> <*type*> **LBRACK** <*exp*> **RBRACK**

<*statements*> -> <*statement*> <*statement\_semicol*>

<*statement\_semicol*> -> <*statement\_semicol*> <*statement*>

<*statement\_semicol*> -> **ε**

<*statement*> -> <*lexp*> **ASSIGN** <*exp*> **SEMICOLON**

<*statement*> -> **RETURN** <*exp*> **SEMICOLON**

<*statement*> -> **NAME** **LPAR** <*pars*> **RPAR**

<*statement*> -> **WRITE** <*exp*> **SEMICOLON**

<*statement*> -> **READ** <*exp*> **SEMICOLON**

<*statement*> -> **IF** **LPAR** <*exp*> **RPAR** <*statement*>

<*statement*> -> **IF** **LPAR** <*exp*> **RPAR** <*statement*> **ELSE** <*statement*>

<*statement*> -> **WHILE** **LPAR** <exp> RPAR <statement>

<*statement*> -> <*block*>

<*lexp*> -> <*var*>

<*lexp*> -> <*lexp*> **LBRACK** <*exp*> **RBRACK**

<*exp*> -> <*lexp*>

<*exp*> -> <*exp*> <*binop*> <*exp*>

<*exp*> -> <*unop*> <*exp*>

<*exp*> -> **LPAR** <*exp*> **RPAR**

<*exp*> -> **NUMBER**

<*exp*> -> **NAME** **LPAR** <*pars*> **RPAR**

<*exp*> -> **QCHAR**

<*exp*> -> **LENGTH** <*exp*>

<*binop*> -> **PLUS**

<*binop*> -> **MINUS**

<*binop*> -> **TIMES**

<*binop*> -> **DIVIDE**

<*binop*> -> **EQUAL**

<*binop*> -> **NEQUAL**

<*binop*> -> **GREATER**

<*binop*> -> **LESS**

<*unop>* -> **NOT**

<*pars*> -> <*exp*> [**COMMA** <*exp*>]\*

<*pars*> -> **ε**

<*var*> -> **NAME**

* 1. **Terminal Symbols (Tokens):**
* **INT** int
* **IF** if
* **ELSE** else
* **NEQUAL** !=
* **RETURN** return
* **LPAR** (
* **RPAR** )
* **LBRACE** {
* **RBRACE** }
* **LBRACK** [
* **RBRACK** ]
* **ASSIGN** =
* **SEMICOLON** ;
* **COMMA** ,
* **PLUS** +
* **MINUS** –
* **TIMES** \*
* **DIVIDE** /
* **EQUAL** ==
* **CHAR** char
* **WRITE** write
* **READ** read
* **GREATER** >
* **LESS** <
* **NOT** !
* **LENGTH** length
* **WHILE** while
* **QCHAR**
* **NUMBER**
* **NAME**
* **COMMENT**

*Remarks:*

* NUME -> a string starting with a letter, followed by 0 or more letters, digits or underscores
* NUMBER -> a string of digits
* QCHAR -> a character between single quotes
* COMMENT -> a string that starts with // and continues until the end of the line
  1. **Non-terminal symbols**
  + <program>
  + <declaration>
  + <fun\_declaration>
  + <formal\_pars>
  + <formal\_par>
  + <block>
  + <var\_declarations>
  + <var\_declaration>
  + <type>
  + <statements>
  + <statement\_semicol>
  + <statement>
  + <lexp>
  + <exp>
  + <binop>
  + <unop>
  + <pars>
  + <var>

1. **The scanner**
   1. **Regular definitions**

* NAME -> [ a – z A – Z ] ( [ a – z A – Z ] | [ 0 – 9 ] | [ \_ ] ) \*
* NUMBRE -> [ 0 – 9 ] +
* QCHAR -> \’ . \’
  1. **Finite automata used for lexical analysis**

The finite automaton for the NAME lexical category:



The finite automaton for the NUMBER lexical category:



The finite automaton for the COMMENT lexical category:



The finite automaton for the QCHAR lexical category:



Deterministic finite automaton for all lexical categories:



The name of some final states in this DFA are:

* State 8: RBRACE
* State 9: LBRACE
* State 10: RPAR
* State 11: LPAR
* State 12: SEMICOLON
* State 13: LBRACK
* State 14: RBRACK
* State 15: COMMA
* State 16: DIVIDE
* State 17: TIMES
* State 18: NOT
* State 19: PLUS
* State 20: MINUS
* State 21: ASSIGN
* State 22: EQUAL
* State 23: NEQUAL
* State 24: GREATER
* State 25: LESS

1. **The parser**

**Eliminating left recursion**

<*program*> -> <*declaration2*>

<*declaration2*> -> <*declaration*> <*declaration2*>

<*declaration*> -> <*type*> **NAME** <*fun\_declaration*>

<*declaration*> -> <*var\_declaration*>

<*fun\_declaration*> -> **LPAR** <*formal\_pars*> **RPAR** <*block*>

<*formal\_pars*> -> <*formal\_par*> <*formal\_pars2*>

<*formal\_pars*> -> **ε**

<*formal\_pars2*> -> **COMMA** <*formal\_par*><*formal\_pars2*>

<*formal\_pars2*> -> **ε**

<*formal\_par*> -> <*type*> **NAME**

<*block*> -> **LBRACE** <*var\_declarations*> <*statements*> **RBRACE**

<*var\_declarations*> -> <*var\_declaration*> <*var\_declarations*>

<*var\_declarations*> -> **ε**

<*var\_declaration* > -> <*type*> **NAME SEMICOLON**

<*var\_declaration* > -> **SEMICOLON**

<*type*> -> **INT** *<type2>*

<*type*> -> **CHAR** *<type2>*

<*statements*> -> <*statement*> <*statements*>

<*statements*> -> **ε**

<*statement*> -> <*lexp*> **ASSIGN** <*exp*> **SEMICOLON**

<*statement*> -> **RETURN** <*exp*> **SEMICOLON**

<*statement*> -> <*lexp*> **LPAR** <*pars*> **RPAR**

<*statement*> -> **WRITE** <*exp*> **SEMICOLON**

<*statement*> -> **READ** <*exp*> **SEMICOLON**

<*statement*> -> **IF** **LPAR** <*exp*> **RPAR** <*statement*>

<*statement*> -> **IF** **LPAR** <*exp*> **RPAR** <*statement*> **ELSE** <*statement*>

<*statement*> -> **WHILE** **LPAR** <exp> **RPAR** *<statement>*

<*statement*> -> <*block*>

<*lexp*> -> <*var*> <*type2*>

<*type2*> -> **LBRACK** <*exp*> **RBRACK** *<type2>*

<*type2*> -> **ε**

*<exp> -> <exp2>*

<*exp*> -> **LPAR** <*exp*> **RPAR**

<*exp*> -> **NUMBER**

<*exp*> -> **NAME** **LPAR** <*pars*> **RPAR**

<*exp*> -> **QCHAR**

<*exp*> -> **LENGTH** <*exp*>

<*exp2*> -> <*lexp*>

<*exp2*> -> <*binop*> <*exp*>

<*exp2*> -> <*unop*> <*exp*>

<*binop*> -> **PLUS**

<*binop*> -> **MINUS**

<*binop*> -> **TIMES**

<*binop*> -> **DIVIDE**

<*binop*> -> **EQUAL**

<*binop*> -> **NEQUAL**

<*binop*> -> **GREATER**

<*binop*> -> **LESS**

<*unop>* -> **NOT**

<*pars*> -> <*exp*> *<pars2>*

<*pars*> -> **ε**

<*pars2*> -> **COMMA** *<exp> <pars2>*

<*pars2*> -> **ε**

<*var*> -> **NAME**

**Non-terminal symbols**

* + <program>
  + <declaration>
  + <declaration2>
  + <fun\_declaration>
  + <formal\_pars>
  + <formal\_pars2>
  + <formal\_par>
  + <block>
  + <var\_declarations>
  + <var\_declaration>
  + <type>
  + <type2>
  + <statements>
  + <statement>
  + <lexp>
  + <exp>
  + <exp2>
  + <binop>
  + <unop>
  + <pars>
  + <pars2>
  + <var>

**The first sets for non-terminal symbols**

**First**(<program>) = **First(**<formal\_par>**) = First(**<formal\_pars>**) = First(**<var\_declaration>**) = First(**<type >**)= First**(<declaration2>) **= {**int, char}

**First**(<declaration>) = {name}

**First**(<fun\_declaration>) = { ‘(‘ }

**First(**<formal\_pars>**) = First(**<var\_declarations>**) = {**int,char, ε**}**

**First(**<formal\_pars2>**) = First(**<pars2>)**= {** ‘,’ , ε**}**

**First(**<block>**) =** { ‘{’ }

**First(**<type2>) = { ‘[‘, ε }

**First**(<statements>) = { ε } U **First**(<statement>)

**First**(<statement>) = {return, write,read,if,while} U **First(**<block>**) U First(**<var**)**

**First**(<exp>) = **First(**<var>**) U {**number, qchar, length**} U First**(<fun\_declaration>) U **First** (<unop>)

**First(**<binop>) = { plus, minus, times, divide, equal, nequal, greater, less }

**First(**<binop2>) = **First**(<exp>)

**First** (<unop>) = {not}

**First(**<pars>) = **First**(<exp>) U { ε }

**First(**<var>) **= First(**<lexp>) **= {**name**}**

**First(**<exp2>) **= First** (<unop>) U **First(**<binop>) U **First(**<lexp>)