Saint-Petersburg State Electro technical University “LETI”

**REPORT**

**Language processors development, laboratory work №2**

**«Syntax analyzer»**

Students: Stepanov K.S.

Ryabikov E.M.

Vasilyev D.A.

FKTI

Group: 9303

Variant 4

Hodirev I.A.

Saint-Petersbourg 2014

Assignment

Goal of this work is to develop a programming language, being set of a language, specified by individual task, and a translator from this language to an intermediate language of type, specified by individual task. Syntax analysis type is also mentioned in individual variant.

Language must support manipulations with variables and constants of specified types and also variables and components of derivative type, which is specified by individual task. Operation set must contain default language operations and operations, specified by individual task. Language must provide type conversion operation in case of types are equivalent. language must provide possibility of defining user types.

Language must provide arithmetic operations with constants, simple variables of default language types, structure type components, parentheses, signs (addition, substraction, division, multiplication) with usual operation priority.

Language must provide using logical expressions with relations, parentheses, and logical operations (and, or, not). If language contains logical constants they must be used too. Operation priority – usual.

Operations with structured type are specified by individual task.

Operators:

* assignment;
* input;
* output;
* complex;
* goto;
* conditional operator with logical expression as condition;
* cycle operator
* operators, specified by individual task

Also program should support comments.

**Variant 4**

*Base language* – Pascal.

*Base types:* Integer, rational, bounded, enumerable.

*Structured type* – Array.

*Rational type operations:* getting numerator and denominator, taking integer and fractional part, conversion to irregular fraction, calculating common denominator, simplification.

*Cycle operator* – with parameter.

*Operation overload* – permitted.

*Type equivalency* – by name.

*Grammar class* – LL(1).

*Intermediate representation*  – tetrad

# Language description

## Language syntax description.

<program>::=**program** <program\_name>;

[<section description>]

**begin**

[<section operators>]

**end.**

< program\_name >::=<id>

< section description >::=[**const** < const description >]

[**type** < types description >]

[**label** < label description >]

[**var** < var description >]< section description >

<описания\_типов>::=<имя\_пользовательского\_типа> = <тип>;

{< types description >}

<user type name>::=<id>

<type>::=<defined\_type>|< user type name >

<defined\_type>::=**Integer**|**Rational**|<array>|<bounded>|<enumerable>

<array>::=**array** [<id\_diap>{,<id\_diap>}] **of** <type>

< id\_diap >::=<unsigned int>..<unsigned int>

< user type name >::=<id>

<bounded>::=<signed int>..<signed int>

<enumerable>::=(<id>{,<id>})

<label description>::=<label>{,<label>};

<label>::=<id>

<const\_description>::=<const name> = <int const>;

{< const\_description >}

< int const >::=<signed int>|<const name>

< signed int >::=[+|-]<unsigned int>

< const name >::=<id>

<var description>::=<variable>{,< variable >} : <type>;

< variable >::=<id>

<operators description>::=<operator>;{< operator >;}

< operator >::=[<label>:]<unlabeled operator >

<unlabeled\_operator>::=<conditional\_operator>|<assignment\_operator>|<cycle\_operator>|<operator\_goto>|<input\_operator>|<output\_operator>|<complex\_operator>

< conditional\_operator >::= **if**<logical\_expression>**then**<operator> [**else**<operator>]

< logical\_expression >::=<disjunction\_operand>{**or**< disjunction\_operand >}

< disjunction\_operand >::=<conjunction\_operand>{**and**< conjunction\_operand >}

< disjunction\_operand >::=[**not**]<negation\_operang>

< negation\_operang >::=(<relation>)|(< logical\_expression >)

< relation >::=< relation operand>{< relation operand >< relation operand >}

< relation operand >::=<expression>

< relation operand >::= >|<|=|<>|<=|>=

<выражение>::=<signed integer>|(+|-)<vector element name>|(+|-)<variable>[.**numerator**|.**denominator**]|(+|-)<signed\_integer>|<operation>

<vector\_element\_name >::=<vactor\_name>[< vector\_element\_name >{,< vector\_element\_number >}]

<vector\_name>::=<id>

< vector\_element\_number >::=<unsigned int>

<operation>::=<operand\_addition>{(+|-) < operand\_addition >}

< operand\_addition >::=< operand\_multiplication>{(\*|/) < operand\_multiplication >}

<operand\_multiplication>::=<specific\_operator>(<operand\_multiplication>)|(<expression>)|<signed integer>|(+|-)< vector\_element\_number >| (+|-)<variable>[.**numerator**|.**denominator**]| (+|-)<integer constant>|<operator\_common\_denomenator>(<expression>,<expression>)

<specific operator>::=**int**|**frac**|**improper**|**simple**

<operator\_common\_denomenator>::=**comden**

<operator\_assignment>::=<variable>:=<expression>

<complex\_operator>::=**begin** <operator>;{<operator>;} **end**

<input\_operator>::=<input\_operator\_name>(<values>)

< input\_operator\_name >::=**read**|**readln**

<values>::=<variable>|<expression>

<output-operator>::=< output\_operator\_name >(<values>)

< output\_operator\_name >::=**write**|**writeln**

<goto\_operator>::=**goto**<label>  
<cycle\_operator>::=**for** <variable>:=<expression> **to** <expression> **do**

<operator>;

<id>::=<letter>|<sign\_underline> {<letter>|< sign\_underline >|<digit>}

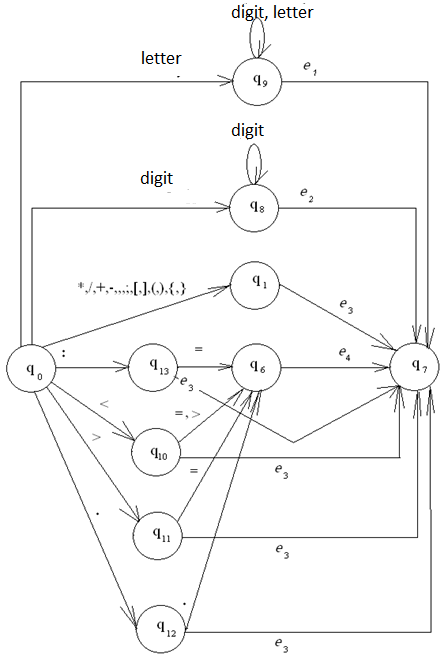
<letter>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<unsigned int>::=<digit>{<digit>}

<digit> ::= 0 **|** 1 **|** 2 **|** 3 **|** 4 **|** 5 **|** 6 **|** 7 **|** 8 **|** 9

< sign\_underline >::= \_

**2. Lexical analysis state machine.**



## 3.Data structures to store lexems

1. Keywords

|  |  |  |
| --- | --- | --- |
| n | token | |
| Lexem type | Lexem value |
| 1 | KARR | array |
| 2 | KOF | of |
| 3 | KBG | begin |
| 4 | KEN | end |
| 5 | KIF | if |
| 6 | KTH | then |
| 7 | KEL | else |
| 8 | KFR | for |
| 9 | KTO | to |
| 10 | KDO | do |
| 11 | KGO | goto |
| 12 | KCON | const |
| 13 | KVAR | var |
| 14 | KLAB | label |
| 15 | KTYP | type |
| 16 | KOR | or |
| 17 | KAND | and |
| 18 | KNOT | not |
| 19 | KRD | read |
| 20 | KRDL | readln |
| 21 | KWT | write |
| 22 | KWTL | writeln |
| 23 | KNUM | numerator |
| 24 | KDEN | denominator |
| 25 | KINT | int |
| 26 | KFRC | frac |
| 27 | KIMP | improper |
| 28 | KSIM | simple |
| 29 | KCDN | comden |
| 30 | KPR | program |

1. Arithmetic operators (OPAR)

|  |  |
| --- | --- |
| 0 | + |
| 1 | - |

1. Multiplication operators(OPMUL)

|  |  |
| --- | --- |
| 0 | \* |
| 1 | / |

1. Diapason operator (OPDPS)

|  |  |
| --- | --- |
| 0 | .. |

1. Assignment operator (OPA)

|  |  |
| --- | --- |
| 0 | := |

1. Initialization operator (OPINIT)

|  |  |
| --- | --- |
| 0 | = |

1. Relation operator (OPCP)

|  |  |
| --- | --- |
| 1 | > |
| 2 | < |
| 3 | <> |
| 4 | <= |
| 5 | >= |

1. point (SP)

|  |  |
| --- | --- |
| 0 | . |

1. colon(SCOL)

|  |  |
| --- | --- |
| 0 | : |

1. Opening parenthesis(SPO)

|  |  |
| --- | --- |
| 0 | ( |

1. Closing parenthesis (SPC)

|  |  |
| --- | --- |
| 0 | ) |

1. Opening bracket(SBKO)

|  |  |
| --- | --- |
| 0 | [ |

1. closing bracket(SBKC)

|  |  |
| --- | --- |
| 0 | ] |

1. Opening brace(SBCO)

|  |  |
| --- | --- |
| 0 | { |

1. Closing brace(SBCC)

|  |  |
| --- | --- |
| 0 | } |

1. comma(SCOM)

|  |  |
| --- | --- |
| 0 | , |

1. semicalon(SSCOL)

|  |  |
| --- | --- |
| 0 | ; |

1. Types

|  |  |  |
| --- | --- | --- |
| ***id*** | ***val*** | ***type*** |
| 0 | Integer | KTINT |
| 1 | Rational | KTRAT |
| … |  |  |

1. identificators(ID)

|  |  |
| --- | --- |
|  | value |
| 0 |  |
| ….. |  |

1. constants(CINT)

|  |  |
| --- | --- |
| type | value |
| 0 | 3 |

**4. Results and tests.**

**Program source:**

program hello;

var a: Integer;

begin

if a = -111 then a := a + a

else b := b + b;

writeln(a.denominator);

end.

**Tokens:**

<?xml version="1.0" encoding="UTF-8"?>

<tokens>

<token id="0" class="keywords">

<type>KPR29</type>

<value>program</value>

</token>

<token id="1" class="ids">

<type>ID0</type>

<value>hello</value>

</token>

<token id="2" class="separators">

<type>SSCOL0</type>

<value>;</value>

</token>

<token id="3" class="keywords">

<type>KVAR12</type>

<value>var</value>

</token>

<token id="4" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="5" class="separators">

<type>SCOL0</type>

<value>:</value>

</token>

<token id="6" class="types">

<type>KTINT0</type>

<value>Integer</value>

</token>

<token id="7" class="separators">

<type>SSCOL0</type>

<value>;</value>

</token>

<token id="8" class="keywords">

<type>KBG2</type>

<value>begin</value>

</token>

<token id="9" class="keywords">

<type>KIF4</type>

<value>if</value>

</token>

<token id="10" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="11" class="separators">

<type>OPINIT0</type>

<value>=</value>

</token>

<token id="12" class="separators">

<type>OPAR1</type>

<value>-</value>

</token>

<token id="13" class="constants">

<type>CINT0</type>

<value>111</value>

</token>

<token id="14" class="keywords">

<type>KTH5</type>

<value>then</value>

</token>

<token id="15" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="16" class="separators">

<type>OPA0</type>

<value>:=</value>

</token>

<token id="17" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="18" class="separators">

<type>OPAR0</type>

<value>+</value>

</token>

<token id="19" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="20" class="keywords">

<type>KEL6</type>

<value>else</value>

</token>

<token id="21" class="ids">

<type>ID2</type>

<value>b</value>

</token>

<token id="22" class="separators">

<type>OPA0</type>

<value>:=</value>

</token>

<token id="23" class="ids">

<type>ID2</type>

<value>b</value>

</token>

<token id="24" class="separators">

<type>OPAR0</type>

<value>+</value>

</token>

<token id="25" class="ids">

<type>ID2</type>

<value>b</value>

</token>

<token id="26" class="separators">

<type>SSCOL0</type>

<value>;</value>

</token>

<token id="27" class="keywords">

<type>KWTL21</type>

<value>writeln</value>

</token>

<token id="28" class="separators">

<type>SPO0</type>

<value>(</value>

</token>

<token id="29" class="ids">

<type>ID1</type>

<value>a</value>

</token>

<token id="30" class="separators">

<type>SP0</type>

<value>.</value>

</token>

<token id="31" class="keywords">

<type>KDEN23</type>

<value>denominator</value>

</token>

<token id="32" class="separators">

<type>SPC0</type>

<value>)</value>

</token>

<token id="33" class="separators">

<type>SSCOL0</type>

<value>;</value>

</token>

<token id="34" class="keywords">

<type>KEN3</type>

<value>end</value>

</token>

<token id="35" class="separators">

<type>SP0</type>

<value>.</value>

</token>

</tokens>

**Tables(dynamic):**

<ids>

<id id="2">

<type>ID</type>

<name>a</name>

</id>

<id id="3">

<type>ID</type>

<name>b</name>

</id>

<id id="1">

<type>ID</type>

<name>hello</name>

</id>

</ids>

<constants>

<constant id="1">

<type>CINT</type>

<name>111</name>

</constant>

</constants>