Lab 1

Problem

1a

Statement: Considering a small programming language (that we shall call mini-language), write 3 small programs in this language.

Deliverables:

p1.* , p2.*, and p3.* and p1err.* - small programs written in your programming language (p1, p2, p3 should be lexically correct; p1err should contain 2 types of lexical errors).

For example:

- p1 and p2: compute de max/min of 3 numbers; verify if a number is prime, compute gcd of 2 numbers, compute the solutions for a 2nd order equation, etc.
- p3: compute the sum of n numbers, computer the max/min of n numbers

2b

The mini-language can be a restricted form of a known programming language, and should contain the following:

- 2 simple data types and a user-defined type
- statements:
  - **assignment**
  - **input/output**
  - **conditional**
  - **loop**
  - **some conditions will be imposed on the way the identifiers and constants can be formed:**
- identifiers: no more than 256 characters
- constants: corresponding to your types

Solution

Lexic.txt

```
Alphabet:
- upper and lower case letters of the English alphabet <letter>
- underline character _
- decimal digits (0-9) <digit>
- operators <operator>
- separators <separator>

Identifiers:
- any combination of letters, or digits that starts with an underscore

Constants:
- integer:
  <non-zero digit> ::= 1 | ... | 9
  <digit> ::= 0 | ... | 9
  <sign> ::= + | -
  <unsigned integer> ::= <non-zero digit> | <unsigned integer> <digit>
  <signed integer> ::= 0 | <unsigned integer> | <sign> <unsigned integer>

- character
  <character literal> := digit | letter
  <character const> := "'" {character literal} "'"

- string
  <character> = <letter> | _ | <digit> | <operator> | <separator>
  <characters> = <character> | <characters> <character>
  <string> := \" {character literal} \"

Special symbols, representing:
- arithmetic operators: + - * / %
- relational operators: = := < <= == => > ?
- separators: () [ ] : ; space ?
- reserved words:
    int char string collection for if while for do elsdo r w
```

token.in

```
+,-,*,/,%
==,:=,<=,<,>,>=,=
(,),[,],?,:,;',',' ','\n',
id const int char string collection for if while for do elsdo r w
```

Syntax.in

<type> ::= int | char | string | collection

<letter> = a | … | z | A | … | Z

<digit> = 0 | … | 9

<identifier> ::= _ | <identifier> <letter> | <identifier> <digit>

<factor> ::= (<expression>) | <identifier> | <constant>

<term operator> ::= * | / | %

<term> ::= <term> <term operator> <factor> | <factor>

<expression operator> ::= + | -

<expression> ::= <expression> <expression operator> <term> | <term> | <ternary expression>

<condition> ::= <expression> <relational operator> <expression>

<ternary expression> ::= <condition> ? <expression> : <expression>

<declaration statement> ::= <type> <identifier>; | <type> <identifier> = <expression>;

<assignment statement> ::= <identifier> = <expression>;

<io statement> ::= r(<identifier>); | w(<identifier>);

<if statement> ::= if (<condition>) do (<statement-list>) | if (<condition>) do (<statement-list>) elsdo (<statement-list>)

<while statement> ::= while (<condition>) do (<statement-list>)

<relational operator> ::= = | < | <= | == | := | => | >

<for statement> ::= for (<statement>, <condition>, <statement>) do (<statement-list>)

<statement> ::= <declaration statement> | <assignment statement> | <io statement> | <if statement> | <while statement> | <for statement>

<statement-list> ::= <statement> | <statement-list> <statement>

<program> ::= null | <statement-list>

p1 - compute gcd of 2 numbers

```
int _a;
int _b;
r(_a);
r(b);

while (_a := _b) do (_a > _b ? _a = _a - _b : _b = _b - _a;)

w(_a);
```

p2 - max of 3 numbers

```
int _a;
int _b;
int _c;
r(_a);
r(_b);
r(_c);

int _max;
_max = _a > _b ? _a : _b;
_max = _max > c ? _max : _c;

w(_max);
```

p3 - compute the sum of n numbers

```
int _n;
r(_n);

int _sum = 0;
while(_n := 0) do (
  int _x;
  r(_x);
  _sum = _sum + _x;
  _n = _n - 1;
)

w(_sum);
```

p1err - computer the max of n numbers

```
int _max = -1;
int n; // n is not a valid identifier: int _n;
r(_n);

whilee(_n := 0) do ( // whilee is not a valid token/reserved word: while;
  int _x;
  r(_x);
  _max = _x > _max ? _x : _max;
  _n = -n - 1;
)

w(_max);
```