Statement: Considering a small programming language (that we shall call mini-langauge), you

have to write a scanner (lexical analyzer).

The mini language should 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.

TODO: a) Mini language Specification

Define the elements of the programming language: reserved words, operators, separators,

identifiers, and constants.

Describe the syntax of the programming language using BNF notation

Example:

Language Specification:

1 .Language Definition:

1.1 Alphabet:

1.1.a. Upper (A-Z) and lower case letters (a-z) of the English alphabet

b. Underline character '\_';

c. Decimal digits (0-9);

d. Space character ' '

e. Symbols . ? !

Lexic:

a.Special symbols, representing:

- operators + - \* / := < <= = >= ^ % = !=

- separators ()[ ] { } : ; space endline tab

- reserved words: int, char, array, do, if, else, while, for, read, write, const, and, or, return, try, catch, finally, true, false

b.identifiers - a sequence of letters and digits and can have an underline between letters and digits, such that the first character is a letter; the rule is:

identifier ::= letter | letter{letter}{digit} | underline letter{letter}{digit}

letter ::= "A" | "B" |...| "Z" | "a" | "b" |...| "z"

digit ::= "0" | "1" |...| "9"

positive ::= "1" |...| "9"

underline::= "\_"

symbol::= "." | "?" | "!"

position::=positive{no}|0|IDENTIFIER

c.constants

1.integer - rule:

nr:=+positive{no}|-positive{no}|positive{no}|0

no:=digit{digit}

2.character

character:='letter'|'digit'|'symbol'

3.string

constchar:="string"

string:=char{char}

char:=letter|digit|symbol

4. bool

bool:=true|false

2.2 Syntax:

The words - predefined tokens are specified between " and ":

a) Sintactical rules:

decllist ::= declaration |declaration ";" decllist

declaration ::= type IDENTIFIER

type1 ::= "bool" | "char" | "int" | "float"

arraydecl ::= "List <" type1 ">" identifier "[" nr "]"

arrayselect::=IDENTIFIER"["(IDENTIFIER|position)"]"

type ::= type1|arraydecl

cmpdstmt ::= "{" stmtlist "}"

stmtlist ::= stmt | stmt ";" stmtlist

stmt ::= simplstmt | structstmt

simplstmt ::= assignstmt | iostmt

assignstmt ::= IDENTIFIER "=" expression | IDENTIFIER "[" position "]" = expression

expression ::= expression "+" term | expression "-" term | term

term ::= term "\*" factor | term "/" factor | term "%" factor | factor

factor ::= "(" expression ")" | IDENTIFIER

iostmt ::= "read" "(" IDENTIFIER ")" ";"| "write" "(" IDENTIFIER ")" ";"

structstmt ::= cmpdstmt | ifstmt | whilestmt | forstmt

ifstmt ::= "if(" condition ")" stmt ["else" stmt]

whilestmt ::= "while(" condition ")" stmt

forstmt ::= "for(" assignstmt ";" condition ";" assignstmt ")" stmt

condition ::= expression RELATION expression

RELATION ::= "<" | "<=" | "==" | ">=" | ">" | "!="

b) Write a small program into your programming language (for example: find the first k prime

numbers, print all prime numbers less than k)

//find the first k prime numbers

int k;

read(k);

int i;

i=2;

int c;

c=0;

while(c!=k)

{

bool ok;

ok=false;

int d;

d=2;

while(d\*d<=i)

{

if(i%d==0)

ok=true;

d=d+1;

}

if(ok==false)

{

write(i);

c=c+1;

}

i=i+1;

}