https://github.com/trutadan/University-Work/tree/main/Semester%205/Formal%20Languages%20and%20Compiler%20Design/lab09

Lexical Analyzer Documentation

The lexical analyzer is responsible for scanning the input source code and identifying various lexical elements, such as constants, identifiers, operators, and reserved words. It performs error handling and provides informative messages for better debugging during the compilation process.

spec.lxi:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int currentLine = 1;
%}

BOOL_CONST       true|false
INT_CONST 0|[+|-]?[1-9][0-9]*
CHAR_CONST       [\'][a-zA-Z0-9][\']
STRING_CONST     [\"][a-zA-Z0-9 ]*[\"]
IDENTIFIER  [a-zA-Z_][a-zA-Z0-9_]*

%%

"array"          {printf("Line %d - reserved word: %s\n", currentLine, yytext); return
ARRAY;}
"bool"           {printf("Line %d - reserved word: %s\n", currentLine, yytext); return
BOOL;}
"string"         {printf("Line %d - reserved word: %s\n", currentLine, yytext); return
STRING;}
"int"            {printf("Line %d - reserved word: %s\n", currentLine, yytext); return INT;}
"char"           {printf("Line %d - reserved word: %s\n", currentLine, yytext); return
CHAR;}
"if"             {printf("Line %d - reserved word: %s\n", currentLine, yytext); return IF;}
"elif"           {printf("Line %d - reserved word: %s\n", currentLine, yytext); return ELIF;}
"else"           {printf("Line %d - reserved word: %s\n", currentLine, yytext); return ELSE;}
"do"             {printf("Line %d - reserved word: %s\n", currentLine, yytext); return DO;}
```

```
"end"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return END;}
"gets"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
GETS;}
"puts"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
PUTS;}
"cuts"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
CUTS;}
"to_i"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return TO_I;}
"start"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
START;}
"stop"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
STOP;}
"nil"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return NIL;}
"and"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return AND;}
"break"			{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
BREAK;}
"or"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return OR;}
"for"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return FOR;}
"while"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return
WHILE;}
"in"		{printf("Line %d - reserved word: %s\n", currentLine, yytext); return IN;}

"+"		{printf("Line %d - operator %s\n", currentLine, yytext); return plus;}
"-"		{printf("Line %d - operator %s\n", currentLine, yytext); return minus;}
"*"		{printf("Line %d - operator %s\n", currentLine, yytext); return mul;}
"/"		{printf("Line %d - operator %s\n", currentLine, yytext); return divs;}
"%"		{printf("Line %d - operator %s\n", currentLine, yytext); return mod;}
"<="		{printf("Line %d - operator %s\n", currentLine, yytext); return lessOrEqual;}
">="		{printf("Line %d - operator %s\n", currentLine, yytext); return
moreOrEqual;}
"<"		{printf("Line %d - operator %s\n", currentLine, yytext); return less;}
">"		{printf("Line %d - operator %s\n", currentLine, yytext); return more;}
"=="		{printf("Line %d - operator %s\n", currentLine, yytext); return equal;}
"!="		{printf("Line %d - operator %s\n", currentLine, yytext); return different;}
"="		{printf("Line %d - operator %s\n", currentLine, yytext); return eq;}

"{"		{printf("Line %d - separator %s\n", currentLine, yytext); return
leftCurlyBracket;}
"}"		{printf("Line %d - separator %s\n", currentLine, yytext); return
rightCurlyBracket;}
```

```
"("             {printf("Line %d - separator %s\n", currentLine, yytext); return
leftRoundBracket;}
")"             {printf("Line %d - separator %s\n", currentLine, yytext); return
rightRoundBracket;}
"["             {printf("Line %d - separator %s\n", currentLine, yytext); return
leftSquareBracket;}
"]"             {printf("Line %d - separator %s\n", currentLine, yytext); return
rightSquareBracket;}
":"             {printf("Line %d - separator %s\n", currentLine, yytext); return colon;}
";"             {printf("Line %d - separator %s\n", currentLine, yytext); return semicolon;}
","             {printf("Line %d - separator %s\n", currentLine, yytext); return comma;}

{BOOL_CONST}    {printf("Line %d - boolean constant: %s\n", currentLine, yytext);
return BOOL_CONST;}
{INT_CONST}     {printf("Line %d - integer constant: %s\n", currentLine, yytext);
return INT_CONST;}
{CHAR_CONST}    {printf("Line %d - character constant: %s\n", currentLine, yytext);
return CHAR_CONST;}
{STRING_CONST}  {printf("Line %d - string constant: %s\n", currentLine, yytext); return
STRING_CONST;}
{IDENTIFIER}    {printf("Line %d - identifier: %s\n", currentLine, yytext); return
IDENTIFIER;}

[ \t]+          {/* Ignore whitespaces */}
[\n]+           {currentLine++;}

[0-9][a-zA-Z0-9_]*                  {printf("@ Line %d - illegal identifier: %s
@\n", currentLine, yytext);}
[+|-]0|[+|-]?0[0-9]+                {printf("@ Line %d - illegal integer
constant: %s @\n", currentLine, yytext);}
[\'][a-zA-Z0-9 ]{2}[\']|[\'][a-zA-Z0-9 ]{2,}[\'] {printf("@ Line %d - illegal character constant:
%s @\n", currentLine, yytext);}
[\"][a-zA-Z0-9_]+|[a-zA-Z0-9_]+[\"]     {printf("@ Line %d - illegal string
constant: %s @\n", currentLine, yytext);}
.                                   {printf("@ Line %d - unrecognized
token: %s @\n", currentLine, yytext);}

%%
```

Parser Documentation

The Bison parser defines the grammar rules for my programming language from the first lab, specifying how different language constructs are syntactically structured. It includes rules for statements, expressions, control flow, and more. The parser collaborates with the lexical analyzer to parse the input source code and provide syntax validation.

spec.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int yylex(void);

void yyerror(char *s);

#define YYDEBUG 1
%}

%union {
    char *str;
}

%token ARRAY
%token BOOL
%token INT
%token CHAR
%token STRING
%token IF
%token ELIF
%token ELSE
%token DO
%token END
%token GETS
%token PUTS
%token CUTS
%token TO_I
%token START
%token STOP
%token WHILE
```

```
%token FOR
%token AND
%token OR
%token IN
%token BREAK
%token TRUE
%token FALSE
%token NIL
%token RETURN

%token plus
%token minus
%token mul
%token divs
%token mod
%token less
%token lessOrEqual
%token equal
%token different
%token moreOrEqual
%token more
%token eq

%token leftRoundBracket
%token rightRoundBracket
%token leftSquareBracket
%token rightSquareBracket
%token leftCurlyBracket
%token rightCurlyBracket
%token comma
%token colon
%token semicolon

%token BOOL_CONST
%token INT_CONST
%token CHAR_CONST
%token STRING_CONST
%token IDENTIFIER

%start program
```

```
%%

const :         BOOL_CONST
                | INT_CONST
                | CHAR_CONST
                | STRING_CONST
                ;

program :       START statement_list STOP
                ;
statement_list :        statement semicolon
                        | statement semicolon statement_list
                        ;
statement :     declaration_statement
                | assignment_statement
                | if_statement
                | while_statement
                | return_statement
                | for_statement
                | break_statement
                | io_statement
                | imod_statement
                ;

declaration_statement : data_type IDENTIFIER
                        ;
data_type :     INT
                | STRING
                | CHAR
                | BOOL
                | ARRAY
                ;

assignment_statement :      IDENTIFIER eq expression
                        | IDENTIFIER eq imod_statement
                        | IDENTIFIER eq array_access
                        | IDENTIFIER eq leftSquareBracket array_values
rightSquareBracket
                        | array_access eq expression
```

```
                ;
array_values :  const
            | array_values comma const
            ;
expression :  expression plus term
            | expression minus term
            | term
            ;
term :  term mul factor
        | term divs factor
        | term mod factor
        | factor
        ;
factor :    leftRoundBracket expression rightRoundBracket
            | const
            | IDENTIFIER
            ;
array_access :  IDENTIFIER leftSquareBracket expression rightSquareBracket
            ;

if_statement :  IF condition statement_list ELIF condition statement_list ELSE
statement_list END
            | IF condition statement_list ELSE statement_list END
            | IF condition statement_list END
            ;
condition :  expression relation expression
            ;
relation :  less
            | lessOrEqual
            | equal
            | different
            | moreOrEqual
            | more
            ;

while_statement :  WHILE condition compound_statement
                ;
compound_statement :  DO statement_list END
                ;
```

```
return_statement :   RETURN expression
                     ;

for_statement :      FOR IDENTIFIER IN expression compound_statement
                     ;

break_statement:     BREAK
                     ;

io_statement :       GETS leftRoundBracket IDENTIFIER rightRoundBracket
                   | PUTS leftRoundBracket IDENTIFIER rightRoundBracket
                   | PUTS leftRoundBracket const rightRoundBracket
                   ;

imod_statement :     CUTS leftRoundBracket IDENTIFIER rightRoundBracket
                     | CUTS leftRoundBracket STRING_CONST rightRoundBracket
                     | TO_I leftRoundBracket IDENTIFIER rightRoundBracket
                     | TO_I leftRoundBracket STRING_CONST rightRoundBracket
                     ;

%%

extern FILE *yyin;

int yywrap() {
        return 1;
}

void yyerror(char *s)
{
        printf("%s\n", s);
}

int main(int argc, char **argv)
{
        if(argc>1) yyin = fopen(argv[1],"r");
        if(argc>2 && !strcmp(argv[2],"-d")) yydebug = 1;
        if(!yyparse()) fprintf(stderr, "\tProgram is syntactically correct.\n");
        return 0;
}
```

Usage:

1. Generate lexical analyzer code:

```
flex spec.lxi
```

2. Generate the parser code:

```
bison -dy spec.y
```

3. Compile the generated code:

```
gcc lex.yy.c y.tab.c -o parser
```

4. Run the parser: parser.exe <source_code_file>

```
parser.exe p1.txt
```

You can see the results of running the parser on the lab1 program files below:

- p1.txt:

```
Line 1 - reserved word: start
Line 2 - reserved word: puts
Line 2 - separator (
Line 2 - string constant: "Give me a natural number"
Line 2 - separator )
Line 2 - separator ;
Line 3 - reserved word: string
Line 3 - identifier: input_string
Line 3 - separator ;
Line 4 - reserved word: gets
Line 4 - separator (
Line 4 - identifier: input_string
Line 4 - separator )
Line 4 - separator ;
Line 5 - reserved word: string
Line 5 - identifier: input_string_without_newline
Line 5 - separator ;
Line 6 - identifier: input_string_without_newline
Line 6 - operator =
Line 6 - reserved word: cuts
Line 6 - separator (
Line 6 - identifier: input_string
Line 6 - separator )
Line 6 - separator ;
Line 7 - reserved word: int
Line 7 - identifier: input_number
Line 7 - separator ;
Line 8 - identifier: input_number
Line 8 - operator =
Line 8 - reserved word: to_i
Line 8 - separator (
Line 8 - identifier: input_string_without_newline
Line 8 - separator )
Line 8 - separator ;
Line 9 - reserved word: bool
Line 9 - identifier: is_prime
Line 9 - separator ;
Line 10 - identifier: is_prime
Line 10 - operator =
Line 10 - boolean constant: true
Line 10 - separator ;
Line 11 - reserved word: int
Line 11 - identifier: number
Line 11 - separator ;
Line 12 - identifier: number
Line 12 - operator =
Line 12 - integer constant: 2
Line 12 - separator ;
Line 13 - reserved word: while
Line 13 - identifier: number
Line 13 - operator <
Line 13 - identifier: input_number
Line 13 - operator /
Line 13 - integer constant: 2
Line 13 - reserved word: do
Line 14 - reserved word: if
Line 14 - identifier: input_number
Line 14 - operator %
Line 14 - identifier: number
Line 14 - operator ==
Line 14 - integer constant: 0
Line 15 - identifier: is_prime
Line 15 - operator =
Line 15 - boolean constant: false
Line 15 - separator ;
Line 16 - reserved word: break
Line 16 - separator ;
Line 17 - reserved word: end
Line 17 - separator ;
Line 18 - reserved word: end
Line 18 - separator ;
Line 19 - reserved word: puts
Line 19 - separator (
Line 19 - identifier: is_prime
```

- p2.txt:

```
Line 1 - reserved word: start
Line 2 - reserved word: puts
Line 2 - separator (
Line 2 - string constant: "Give me a natural number"
Line 2 - separator )
Line 2 - separator ;
Line 3 - reserved word: string
Line 3 - identifier: input_string
Line 3 - separator ;
Line 4 - reserved word: gets
Line 4 - separator (
Line 4 - identifier: input_string
Line 4 - separator )
Line 4 - separator ;
Line 5 - reserved word: string
Line 5 - identifier: input_string_without_newline
Line 5 - separator ;
Line 6 - identifier: input_string_without_newline
Line 6 - operator =
Line 6 - reserved word: cuts
Line 6 - separator (
Line 6 - identifier: input_string
Line 6 - separator )
Line 6 - separator ;
Line 7 - reserved word: int
Line 7 - identifier: a
Line 7 - separator ;
Line 8 - identifier: a
Line 8 - operator =
Line 8 - reserved word: to_i
Line 8 - separator (
Line 8 - identifier: input_string_without_newline
Line 8 - separator )
Line 8 - separator ;
Line 9 - reserved word: puts
Line 9 - separator (
Line 9 - string constant: "Give me another natural number"
Line 9 - separator )
Line 9 - separator ;
Line 10 - reserved word: gets
Line 10 - separator (
Line 10 - identifier: input_string
Line 10 - separator )
Line 10 - separator ;
Line 11 - identifier: input_string_without_newline
Line 11 - operator =
Line 11 - reserved word: cuts
Line 11 - separator (
Line 11 - identifier: input_string
Line 11 - separator )
Line 11 - separator ;
Line 12 - reserved word: int
Line 12 - identifier: b
Line 12 - separator ;
Line 13 - identifier: b
Line 13 - operator =
Line 13 - reserved word: to_i
Line 13 - separator (
Line 13 - identifier: input_string_without_newline
Line 13 - separator )
Line 13 - separator ;
Line 14 - reserved word: if
Line 14 - identifier: a
Line 14 - operator <
Line 14 - identifier: b
Line 15 - reserved word: int
Line 15 - identifier: aux
Line 15 - separator ;
Line 16 - identifier: aux
Line 16 - operator =
Line 16 - identifier: a
Line 16 - separator ;
Line 17 - identifier: a
Line 17 - operator =
Line 17 - identifier: b
Line 17 - separator ;
Line 18 - identifier: b
Line 18 - operator =
Line 18 - identifier: aux
Line 18 - separator ;
Line 19 - reserved word: end
Line 19 - separator ;
Line 20 - reserved word: while
Line 20 - identifier: b
Line 20 - operator !=
Line 20 - integer constant: 0
Line 20 - reserved word: do
Line 21 - reserved word: int
Line 21 - identifier: aux
Line 21 - separator ;
Line 22 - identifier: aux
Line 22 - operator =
Line 22 - identifier: b
Line 22 - separator ;
Line 23 - identifier: b
Line 23 - operator =
Line 23 - identifier: a
Line 23 - operator %
Line 23 - identifier: b
Line 23 - separator ;
Line 24 - identifier: a
Line 24 - operator =
Line 24 - identifier: aux
Line 24 - separator ;
Line 25 - reserved word: end
Line 25 - separator ;
Line 26 - reserved word: puts
Line 26 - separator (
Line 26 - identifier: a
Line 26 - separator )
Line 26 - separator ;
Line 27 - reserved word: stop
       Program is syntactically correct.
```

- p3.txt:

```
Line 1 - reserved word: start
Line 2 - reserved word: array
Line 2 - identifier: arr
Line 2 - separator ;
Line 3 - identifier: arr
Line 3 - operator =
Line 3 - separator [
Line 3 - integer constant: 3
Line 3 - separator ,
Line 3 - integer constant: 1
Line 3 - separator ,
Line 3 - integer constant: 10
Line 3 - separator ,
Line 3 - integer constant: 4
Line 3 - separator ,
Line 3 - integer constant: -6
Line 3 - separator ,
Line 3 - integer constant: 12
Line 3 - separator ,
Line 3 - integer constant: 4
Line 3 - separator ,
Line 3 - integer constant: 2
Line 3 - separator ,
Line 3 - integer constant: -4
Line 3 - separator ]
Line 3 - separator ;
Line 4 - reserved word: int
Line 4 - identifier: max
Line 4 - separator ;
Line 5 - identifier: max
Line 5 - operator =
Line 5 - identifier: arr
Line 5 - separator [
Line 5 - integer constant: 0
Line 5 - separator ]
Line 5 - separator ;
Line 6 - reserved word: for
Line 6 - identifier: number
Line 6 - reserved word: in
Line 6 - identifier: arr
Line 6 - reserved word: do
Line 7 - reserved word: if
Line 7 - identifier: number
Line 7 - operator >
Line 7 - identifier: max
Line 8 - identifier: max
Line 8 - operator =
Line 8 - identifier: number
Line 8 - separator ;
Line 9 - reserved word: end
Line 9 - separator ;
Line 10 - reserved word: end
Line 10 - separator ;
Line 11 - reserved word: puts
Line 11 - separator (
Line 11 - identifier: max
Line 11 - separator )
Line 11 - separator ;
Line 12 - reserved word: stop
        Program is syntactically correct.
```

- p1err.txt:

```
Line 1 - reserved word: start
Line 2 - reserved word: puts
Line 2 - separator (
Line 2 - string constant: "Give me a natural number"
Line 2 - separator )
Line 2 - separator ;
Line 3 - reserved word: string
Line 3 - identifier: input_string
Line 3 - separator ;
Line 4 - reserved word: gets
Line 4 - separator (
Line 4 - identifier: input_string
Line 4 - separator )
Line 4 - separator ;
Line 5 - reserved word: string
Line 5 - identifier: input_string_without_newline
Line 5 - separator ;
Line 6 - identifier: input_string_without_newline
Line 6 - operator =
Line 6 - reserved word: cuts
Line 6 - separator (
Line 6 - identifier: input_string
Line 6 - separator )
Line 6 - separator ;
Line 7 - reserved word: int
Line 7 - identifier: input_number
Line 7 - separator ;
Line 8 - identifier: input_number
Line 8 - operator =
Line 8 - reserved word: to_i
Line 8 - separator (
Line 8 - identifier: input_string_without_newline
Line 8 - separator )
Line 8 - separator ;
Line 9 - reserved word: bool
@ Line 9 - illegal identifier: 1s_prime @
Line 9 - separator ;
syntax error
```