Lex-Yacc Laboratory

Popa Alex Ovidiu, 936/1

https://github.com/alexovidiupopa/flcd/tree/main/lex-yacc

Commands:

lex specif.lxi

yacc –d lang.y

gcc lex.yy.c y.tab.c -o exe –lfl

./exe < p1.txt

Specif.lxi:

```
%{
#include <stdio.h>
#include <string.h>
int lines = 0;
%}

%option noyywrap
%option caseless

DIGIT           [0-9]
WORD            \"[a-zA-Z0-9]*\"
NUMBER                  [+-]?[1-9][0-9]*|0$
CHARACTER       \'[a-zA-Z0-9]\'
CONST           {WORD}|{NUMBER}|{CHARACTER}
ID              [a-zA-Z][a-zA-Z0-9_]{0,7}
```

```
%%

and {printf("Reserved word: %s\n", yytext);}
array   {printf( "Reserved word: %s\n", yytext);}
else    {printf( "Reserved word: %s\n", yytext);}
for     {printf( "Reserved word: %s\n", yytext);}
go      {printf( "Reserved word: %s\n", yytext);}
if      {printf( "Reserved word: %s\n", yytext);}
number          {printf( "Reserved word: %s\n", yytext);}
or      {printf( "Reserved word: %s\n", yytext);}
cin     {printf( "Reserved word: %s\n", yytext);}
cout    {printf( "Reserved word: %s\n", yytext);}
string  {printf( "Reserved word: %s\n", yytext);}
while   {printf( "Reserved word: %s\n", yytext);}
xor     {printf( "Reserved word: %s\n", yytext);}

{ID}    {printf( "Identifier: %s\n", yytext );}

{CONST}         {printf( "Constant: %s\n", yytext );}

":"     {printf( "Separator: %s\n", yytext );}
";"     {printf( "Separator: %s\n", yytext );}
","     {printf( "Separator: %s\n", yytext );}
"."     {printf( "Separator: %s\n", yytext );}
"{"     {printf( "Separator: %s\n", yytext );}
"}"     {printf( "Separator: %s\n", yytext );}
"("     {printf( "Separator: %s\n", yytext );}
```

```
")"       {printf( "Separator: %s\n", yytext );}
"["       {printf( "Separator: %s\n", yytext );}
"]"       {printf( "Separator: %s\n", yytext );}
"+"       {printf( "Operator: %s\n", yytext );}
"-"       {printf( "Operator: %s\n", yytext );}
"*"       {printf( "Operator: %s\n", yytext );}
"/"       {printf( "Operator: %s\n", yytext );}
"<"       {printf( "Operator: %s\n", yytext );}
">"       {printf( "Operator: %s\n", yytext );}
"<="      {printf( "Operator: %s\n", yytext );}
">="      {printf( "Operator: %s\n", yytext );}
"!="      {printf( "Operator: %s\n", yytext );}
"=="      {printf( "Operator: %s\n", yytext );}
"="       {printf( "Separator: %s\n", yytext );}
"!"       {printf( "Operator: %s\n", yytext );}
">>"      {printf( "Operator: %s\n", yytext );}
"<<"      {printf( "Operator: %s\n", yytext );}



[ \t]+    {}
[\n]+ {lines++;}

[+-]?0[0-9]* {printf("Illegal constant at line %d\n", lines);}

[a-zA-Z][a-zA-Z0-9]{8,}{printf("Illegal size of the identifier at line %d\n", lines);}
```

```
[0-9~@#$%^][a-zA-Z0-9]{0,7}{printf("Illegal identifier at line %d\n", lines);}

\'[a-zA-Z0-9]{2,}\' {printf("Character of length >=2 at line %d\n", lines);}

%%
```

Lang.y

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

#define YYDEBUG 1
%}

%token AND
%token ARRAY
%token ELSE
%token FOR
%token GO
%token IF
%token NUMBER
%token OR
%token CIN
%token COUT
%token STRING
%token WHILE
%token XOR
```

```
%token ID
%token CONST

%token ATRIB
%token EQ
%token NE
%token LE
%token GE
%token LT
%token GT
%token NOT

%token DOT

%left '+' '-' '*' '/'

%token PLUS
%token MINUS
%token DIV
%token MOD
%token MUL

%token OPEN_CURLY_BRACKET
%token CLOSED_CURLY_BRACKET
%token OPEN_ROUND_BRACKET
%token CLOSED_ROUND_BRACKET
```

```
%token OPEN_RIGHT_BRACKET
%token CLOSED_RIGHT_BRACKET

%token READ_OP
%token WRITE_OP

%token COMMA
%token SEMI_COLON
%token COLON
%token SPACE

%start program

%%
program : GO cmpdstmt
;
declaration :  type ID
   ;
type :  NUMBER | STRING | typeTemp
  ;
typeTemp : /*Empty*/ | ARRAY OPEN_RIGHT_BRACKET CONST CLOSED_RIGHT_BRACKET
 ;
cmpdstmt : OPEN_CURLY_BRACKET stmtlist CLOSED_CURLY_BRACKET
 ;
stmtlist :  stmt stmtTemp
 ;
stmtTemp : /*Empty*/ | stmtlist
```

;

stmt : simplstmt SEMI_COLON | structstmt

;

simplstmt : assignstmt | iostmt | declaration

;

structstmt : cmpdstmt | ifstmt | whilestmt | forstmt

;

ifstmt : IF boolean_condition cmpdstmt tempIf

;

tempIf : /*Empty*/ | ELSE cmpdstmt

;

forstmt : FOR forheader cmpdstmt

;

forheader : OPEN_ROUND_BRACKET NUMBER assignstmt SEMI_COLON boolean_condition SEMI_COLON assignstmt CLOSED_ROUND_BRACKET

;

whilestmt : WHILE boolean_condition cmpdstmt

;

assignstmt : ID ATRIB expression

;

expression : arithmetic2 arithmetic1

;

arithmetic1 : PLUS arithmetic2 arithmetic1 | MINUS arithmetic2 arithmetic1 | /*Empty*/

;

arithmetic2 : multiply2 multiply1

;

multiply1 : MUL multiply2 multiply1 | DIV multiply2 multiply1 | /*Empty*/

```
 ;

multiply2 : OPEN_ROUND_BRACKET expression CLOSED_ROUND_BRACKET | CONST | ID |
IndexedIdentifier

 ;

IndexedIdentifier :  ID OPEN_RIGHT_BRACKET CONST CLOSED_RIGHT_BRACKET

 ;

iostmt :  CIN READ_OP ID | COUT WRITE_OP ID | COUT WRITE_OP CONST

    ;

condition : OPEN_ROUND_BRACKET expression GT expression CLOSED_ROUND_BRACKET |

 OPEN_ROUND_BRACKET expression GE expression CLOSED_ROUND_BRACKET |

 OPEN_ROUND_BRACKET expression LT expression CLOSED_ROUND_BRACKET |

 OPEN_ROUND_BRACKET expression LE expression CLOSED_ROUND_BRACKET |

 OPEN_ROUND_BRACKET expression EQ expression CLOSED_ROUND_BRACKET |

 OPEN_ROUND_BRACKET expression NE expression CLOSED_ROUND_BRACKET

 ;

boolean_condition : OPEN_ROUND_BRACKET condition boolean_cond_temp
CLOSED_ROUND_BRACKET

 ;

boolean_cond_temp : /*Empty*/ | AND boolean_condition | OR boolean_condition | XOR
boolean_condition

 ;

%%

yyerror(char *s)

{

printf("%s\n",s);

}


extern FILE *yyin;
```

```
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, "\tO.K.\n");
}
```

P1.txt

```
go{
number a;
number b;
number c;
a=+2;
a=-10;
b=+0;
c=154;
number bCBCA123;
number acb;
number 123abc;
cin>>a;
cin>>b;
cin>>c;
number max ;
```

```
if(a>b and a>c){
max = a;
}
if(b>a and b>c){
max=b;
}
if(c>a and c>b){
max=c;
}
cout<<max;
cout<<"ok";
cout<<"alex;
cout<<'asd';
}
```