Lab9 - yacc

> Github link: https://github.com/DiaconuAna/Formal-Languages-and-Compiler-Design/tree/main/Lab9 - yacc

▼ lang2.lxi - with tokens

```
%option noyywrap

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

DIGIT     [0-9]
WORD    \"[a-zA-Z0-9]*\"
INTEGER   [+-]?[1-9][0-9]*|0
CHARACTER   \'[a-zA-Z0-9]\'
constant  {WORD}|{INTEGER}|{CHARACTER}
identifier  [a-zA-Z][a-zA-Z0-9]*

%%

in           {printf( "Reserved word: %s\n", yytext); return IN;}
out          {printf( "Reserved word: %s\n", yytext); return OUT;}
number       {printf( "Reserved word: %s\n", yytext); return NUMBER;}
begin        {printf( "Reserved word: %s\n", yytext); return IDK;}
string       {printf( "Reserved word: %s\n", yytext); return STRING;}
character    {printf( "Reserved word: %s\n", yytext); return CHARACTER;}
if         {printf( "Reserved word: %s\n", yytext); return IF;}
end          {printf( "Reserved word: %s\n", yytext); return END;}
end_if       {printf( "Reserved word: %s\n", yytext); return END_IF;}
end_for      {printf( "Reserved word: %s\n", yytext); return END_FOR;}
while        {printf( "Reserved word: %s\n", yytext); return WHILE;}
for        {printf( "Reserved word: %s\n", yytext); return FOR;}
end_while    {printf( "Reserved word: %s\n", yytext); return END_WHILE;}
else         {printf( "Reserved word: %s\n", yytext); return ELSE;}

","        {printf( "Separator: %s\n", yytext ); return COMMA;}
";"        {printf( "Separator: %s\n", yytext ); return SEMI_COLON;}
":"        {printf( "Separator: %s\n", yytext ); return COLON;}
"("        {printf( "Separator: %s\n", yytext ); return OPEN_CURLY_BRACKET;}
")"        {printf( "Separator: %s\n", yytext ); return CLOSED_CURLY_BRACKET;}
"["        {printf( "Separator: %s\n", yytext ); return OPEN_RIGHT_BRACKET;}
"]"        {printf( "Separator: %s\n", yytext ); return CLOSED_RIGHT_BRACKET;}
env
"<-"     {printf( "Assignment Operator: %s\n", yytext ); return ASSIGNMENT;}
"+"        {printf( "Operator: %s\n", yytext ); return ADD;}
"-"        {printf( "Operator: %s\n", yytext ); return SUB;}
"*"        {printf( "Operator: %s\n", yytext ); return MUL;}
"/"        {printf( "Operator: %s\n", yytext ); return DIV;}
"mod"    {printf( "Operator: %s\n", yytext ); return MOD;}
"="        {printf( "Operator: %s\n", yytext ); return EQ;}
"<"        {printf( "Operator: %s\n", yytext ); return LT;}
"<="     {printf( "Operator: %s\n", yytext ); return LTE;}
">"        {printf( "Operator: %s\n", yytext ); return GT;}
">="     {printf( "Operator: %s\n", yytext ); return GTE;}
"<>"     {printf( "Operator: %s\n", yytext ); return NE;}
"and"    {printf( "Operator: %s\n", yytext ); return AND;}
"or"     {printf( "Operator: %s\n", yytext ); return OR;}
"not"    {printf( "Operator: %s\n", yytext ); return NOT;}

{identifier}  {printf( "Identifier: %s\n", yytext); return IDENTIFIER;}
{constant}      {printf( "Constant: %s\n", yytext ); return CONSTANT;}

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

[+-]?0[0-9]*                  {printf("Illegal integer at line\n"); return -1;}
[0-9]+[a-zA-Z_]+[a-zA-Z0-9_]*   {printf("Illegal identifier\n"); return -1;}
\'[a-zA-Z0-9]{2,}\'             {printf("Character of length >=2 at line\n"); return -1;}
.                             {printf("Lexical error\n"); return -1;}

%%
```

▼ lang.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#define YYDEBUG 1

%}

%token IN
%token OUT
%token IDK
%token NUMBER
%token STRING
%token CHARACTER
%token IF
%token END
%token END_IF
%token END_FOR
%token WHILE
%token END_WHILE
%token ELSE
%token FOR

%token ASSIGNMENT
%token EQ
%token LT
%token LTE
%token GT
%token GTE
%token NE
%token AND
%token OR
%token NOT

%token IDENTIFIER
%token CONSTANT

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

%token ADD
%token SUB
%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 COMMA
%token SEMI_COLON
%token COLON

%start program
%error-verbose

%%

program : IDK COLON statement_list END
    ;
statement_list : statement | statement statement_list
         ;
statement : simple_statement | if_stmt | while_stmt | for_stmt
      ;
simple_statement : declaration | io_stmt | assignment_stmt
         ;
declaration : type IDENTIFIER SEMI_COLON
      ;
type : simple_type
  ;
simple_type : NUMBER | STRING | CHARACTER
      ;
io_stmt : IN IDENTIFIER SEMI_COLON | OUT IDENTIFIER SEMI_COLON | OUT CONSTANT SEMI_COLON
    ;
assignment_stmt : IDENTIFIER ASSIGNMENT expression SEMI_COLON
        ;
expression : expression ADD term | expression SUB term | term
       ;
term : term MUL factor | term DIV factor | term MOD factor | factor
   ;
factor : OPEN_CURLY_BRACKET expression CLOSED_CURLY_BRACKET | IDENTIFIER | CONSTANT
     ;
if_stmt : IF condition COLON statement_list END_IF | IF condition COLON statement_list ELSE statement_list END_IF
```

```
    ;
condition : OPEN_CURLY_BRACKET expression relation expression CLOSED_CURLY_BRACKET | NOT OPEN_CURLY_BRACKET expression CLOSED_CURL
      ;
relation : EQ | NE | LT | LTE | GT | GTE | AND | OR | NOT
     ;
while_stmt : WHILE condition COLON statement_list END_WHILE
       ;
for_stmt : FOR IDENTIFIER ASSIGNMENT CONSTANT COMMA IDENTIFIER COMMA CONSTANT COLON statement_list END_FOR
     ;
%%

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

```
begin:
number a;
number b;
number c;
number min;
in a;
in b;
in c;
min <- a;
if (b < min):
min <- b;
end_if
if (c < min):
min <- c;
end_if
out min;
end
```

▼ p1_out.txt

```
Reserved word: begin
Separator: :
Reserved word: number
Identifier: a
Separator: ;
Reserved word: number
Identifier: b
Separator: ;
Reserved word: number
Identifier: c
Separator: ;
Reserved word: number
Identifier: min
Separator: ;
Reserved word: in
Identifier: a
Separator: ;
Reserved word: in
Identifier: b
Separator: ;
Reserved word: in
Identifier: c
Separator: ;
Identifier: min
Assignment Operator: <-
Identifier: a
Separator: ;
Reserved word: if
Separator: (
Identifier: b
Operator: <
Identifier: min
Separator: )
Separator: :
```

```
Identifier: min
Assignment Operator: <-
Identifier: b
Separator: ;
Reserved word: end_if
Reserved word: if
Separator: (
Identifier: c
Operator: <
Identifier: min
Separator: )
Separator: :
Identifier: min
Assignment Operator: <-
Identifier: c
Separator: ;
Reserved word: end_if
Reserved word: out
Identifier: min
Separator: ;
Reserved word: end
```

▼ p2.txt

```
begin:
number b;
number div;

  in a;



  for div<-1,a,1:
    if (div mod a = 0):
      out div;
    end_if
  end_for
end
```

▼ p2_out.txt

```
Reserved word: begin
Separator: :
Reserved word: number
Identifier: b
Separator: ;
Reserved word: number
Identifier: div
Separator: ;
Reserved word: in
Identifier: a
Separator: ;
Reserved word: for
Identifier: div
Assignment Operator: <-
Constant: 1
Separator: ,
Identifier: a
Separator: ,
Constant: 1
Separator: :
Reserved word: if
Separator: (
Identifier: div
Operator: mod
Identifier: a
Operator: =
Constant: 0
Separator: )
Separator: :
Reserved word: out
Identifier: div
Separator: ;
Reserved word: end_if
Reserved word: end_for
Reserved word: end
```