lex-yacc Documentation

Wednesday, January 11, 2023 9:38 AM

Commands:

```
flex scanner.lxi
bison  parser.y
gcc lex.yy.c parser.tab.c -o a.exe
./a.exe < P1.txt
```

Scanner.lxi

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

%option noyywrap

IDENTIFIER          [a-zA-Z][a-zA-Z0-9_]*
NUMBER_CONST        0|[+|-]?[1-9][0-9]*([.][0-9]*)?|[+|-]?0[.][0-9]*
STRING_CONST [\"][a-zA-Z0-9_]*[\"]
CHAR_CONST          [\'][a-zA-Z0-9_][\']

%%

"int"          {printf("Reserved word: %s\n", yytext);return INT;}
"float"        {printf("Reserved word: %s\n", yytext);return FLOAT;}
"long"         {printf("Reserved word: %s\n", yytext);return LONG;}
"unsigned"     {printf("Reserved word: %s\n", yytext);return UNSIGNED;}
"string" {printf("Reserved word: %s\n", yytext);return STRING;}
"char"         {printf("Reserved word: %s\n", yytext);return CHAR;}
"while"        {printf("Reserved word: %s\n", yytext);return WHILE;}
"if"           {printf("Reserved word: %s\n", yytext);return IF;}
"else"         {printf("Reserved word: %s\n", yytext);return ELSE;}
"read"         {printf("Reserved word: %s\n", yytext);return READ;}
"print"        {printf("Reserved word: %s\n", yytext);return PRINT;}

"+"                 {printf("Operator: %s\n", yytext);return plus;}
"-"                 {printf("Operator: %s\n", yytext);return minus;}
"*"                 {printf("Operator: %s\n", yytext);return mul;}
"/"                 {printf("Operator: %s\n", yytext);return division;}
"%"                 {printf("Operator: %s\n", yytext);return mod;}
"="                 {printf("Operator: %s\n", yytext);return eq;}
"=="           {printf("Operator: %s\n", yytext);return equal;}
"!="           {printf("Operator: %s\n", yytext);return different;}
"<"                 {printf("Operator: %s\n", yytext);return less;}
```

```
">"                             {printf("Operator: %s\n", yytext);return more;}
"<="                    {printf("Operator: %s\n", yytext);return lessOrEqual;}
">="                    {printf("Operator: %s\n", yytext);return moreOrEqual;}

"("                             {printf("Separator: %s\n", yytext);return leftRoundBracket;}
")"                             {printf("Separator: %s\n", yytext);return rightRoundBracket;}
";"                             {printf("Separator: %s\n", yytext);return semicolon;}
"{"                             {printf("Separator: %s\n", yytext);return leftCurlyBracket;}
"}"                             {printf("Separator: %s\n", yytext);return rightCurlyBracket;}

{IDENTIFIER}                    {printf("Identifier: %s\n", yytext);return IDENTIFIER;}
{NUMBER_CONST}                          {printf("Number: %s\n", yytext);return NUMBER_CONST;}
{STRING_CONST}                          {printf("String: %s\n", yytext);return STRING_CONST;}
{CHAR_CONST}                    {printf("Character: %s\n", yytext);return CHAR_CONST;}

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

[0-9_][a-zA-Z0-9_]*                     {printf("Illegal identifier at line %d\n", currentLine); return -1;}
[+|-]0                  {printf("Illegal numeric constant at line %d\n", currentLine); return -1;}
[+|-]?[0][0-9]*([.][0-9]*)?                             {printf("Illegal numeric constant at line %d\n", currentLine);
return -1;}
[\'][a-zA-Z0-9_]{2,}[\']|[\'][a-zA-Z0-9_]|[a-zA-Z0-9_][\']                      {printf("Illegal character constant at line
%d\n", currentLine); return -1;}
[\"][a-zA-Z0-9_]+|[a-zA-Z0-9_]+[\"]                     {printf("Illegal string constant at line %d\n",
currentLine); return -1;}

%%
```

Parser.y

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

#define YYDEBUG 1
%}

%token INT
%token FLOAT
%token LONG
%token UNSIGNED
%token STRING
%token CHAR
%token WHILE
%token IF
%token ELSE
%token READ
%token PRINT

%token plus
%token minus
```

```
%token mul
%token division
%token mod
%token eq
%token equal
%token different
%token less
%token more
%token lessOrEqual
%token moreOrEqual

%token leftRoundBracket
%token rightRoundBracket
%token semicolon
%token leftCurlyBracket
%token rightCurlyBracket

%token IDENTIFIER
%token NUMBER_CONST
%token STRING_CONST
%token CHAR_CONST

%start program

%%

program : declaration_list statements
declaration_list : declaration declaration_list | /*Empty*/
declaration : var_type IDENTIFIER equal_expression semicolon
equal_expression : eq expression | /*Empty*/
var_type : INT | FLOAT | LONG | UNSIGNED | CHAR | STRING
expression : term sign_and_expression
sign_and_expression : sign expression | /*Empty*/
sign : plus | minus | mul | division | mod
term : IDENTIFIER | constant
constant : NUMBER_CONST | STRING_CONST | CHAR_CONST
statements : statement statements | /*Empty*/
statement : simple_stmt | struct_stmt
simple_stmt : assignment_stmt | input_output_stmt
struct_stmt : if_stmt | while_stmt
assignment_stmt : IDENTIFIER eq expression semicolon
input_output_stmt : READ leftRoundBracket term rightRoundBracket semicolon | PRINT
leftRoundBracket term rightRoundBracket semicolon
if_stmt : IF leftRoundBracket condition rightRoundBracket leftCurlyBracket statements rightCurlyBracket
else_stmt
else_stmt : ELSE leftCurlyBracket statements rightCurlyBracket | /*Empty*/
while_stmt : WHILE leftRoundBracket condition rightRoundBracket leftCurlyBracket statements
rightCurlyBracket
condition : expression relation expression
relation : equal | different | less | more | lessOrEqual | moreOrEqual

%%
```

```
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, "\tProgram is syntactically correct.\n");
}
```

P2.txt

```
int a=-3.7*5;
string b="string";
char c='x';
read(a);
read(b);
while (b!=0) {
	c=a%b;
	a=b;
	b=c;
}
if(a==0){
	print(a);
}
else{
	print(b);
}
```