Commands

flex lang.lxi

bison -d lang.y

gcc lang.tab.c lex.yy.c -o result

lang.lxi

%{

#include <stdio.h>

#include <string.h>

#include "lang.tab.h"

int lines = 0;

%}

%option noyywrap

%option caseless

DIGIT [0-9]

WORD \"[a-zA-Z0-9]\*\"

INTEGER [+-]?[1-9][0-9]\*

CHARACTER \'[a-zA-Z0-9]\'

constant {WORD}|{INTEGER}|{CHARACTER}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

read {printf( "Reserved word: %s\n", yytext); return READ;}

write {printf( "Reserved word: %s\n", yytext); return WRITE;}

if {printf( "Reserved word: %s\n", yytext); return IF;}

else {printf( "Reserved word: %s\n", yytext); return ELSE;}

for {printf( "Reserved word: %s\n", yytext); return FOR;}

while {printf( "Reserved word: %s\n", yytext); return WHILE;}

break {printf( "Reserved word: %s\n", yytext); return BREAK;}

integer {printf( "Reserved word: %s\n", yytext); return INTEGER;}

string {printf( "Reserved word: %s\n", yytext); return STRING;}

character {printf( "Reserved word: %s\n", yytext); return CHARACTER;}

array {printf( "Reserved word: %s\n", yytext); return ARRAY;}

return {printf( "Reserved word: %s\n", yytext); return RETURN;}

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

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

";" {printf( "Separator: %s\n", yytext ); return SEMI\_COLON;}

"," {printf( "Separator: %s\n", yytext ); return COMMA;}

"{" {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\_ROUND\_BRACKET;}

")" {printf( "Separator: %s\n", yytext ); return CLOSED\_ROUND\_BRACKET;}

"[" {printf( "Separator: %s\n", yytext ); return OPEN\_RIGHT\_BRACKET;}

"]" {printf( "Separator: %s\n", yytext ); return CLOSED\_RIGHT\_BRACKET;}

"+" {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;}

"<" {printf( "Operator: %s\n", yytext ); return LT;}

">" {printf( "Operator: %s\n", yytext ); return GT;}

"!=" {printf( "Operator: %s\n", yytext ); return NE;}

"==" {printf( "Operator: %s\n", yytext ); return EQ;}

"=" {printf( "Separator: %s\n", yytext ); return ASIGN;}

"!" {printf( "Operator: %s\n", yytext ); return NOT;}

[ \t]+ {}

[\n]+ {lines++;}

[+-]?0[0-9]\* {printf("Illegal integer at line %d\n", lines); return -1;}

[0-9]+[a-zA-Z\_]+[a-zA-Z0-9\_]\* {printf("Illegal identifier %d\n", lines); return -1;}

\'[a-zA-Z0-9]{2,}\' {printf("Character of length >=2 at line %d\n", lines); return -1;}

%%

lang.y

%{

#include <stdio.h>

#include <stdlib.h>

#define YYDEBUG 1

%}

%token READ

%token WRITE

%token IF

%token ELSE

%token FOR

%token WHILE

%token BREAK

%token INTEGER

%token STRING

%token CHARACTER

%token ARRAY

%token RETURN

%token IDENTIFIER

%token CONSTANT

%token ATRIB

%token EQ

%token NE

%token LT

%token LE

%token GT

%token GE

%token NOT

%token ASIGN

%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 SPACE

%%

program : START compoundStatement

;

declaration : type SPACE IDENTIFIER

;

simpleType : INTEGER | STRING | CHARACTER

;

arrayDeclaration : simpleType SPACE ARRAY OPEN\_RIGHT\_BRACKET CONSTANT CLOSED\_RIGHT\_BRACKET

;

type : simpleType | arrayDeclaration

;

compoundStatement : OPEN\_CURLY\_BRACKET statementList CLOSED\_CURLY\_BRACKET

;

statementList : statement | statement SEMI\_COLON statement

;

statement : simpleStatement | structStatement

;

simpleStatement : assignStatement | ioStatement | declaration

;

structStatement : compoundStatement | ifStatement | whileStatement | forStatement

;

ifStatement : IF condition statement ELSE statement

;

forStatement : FOR OPEN\_ROUND\_BRACKET INTEGER assignStatement SEMI\_COLON condition SEMI\_COLON assignStatement CLOSED\_ROUND\_BRACKET statement

;

whileStatement : WHILE condition statement

;

assignStatement : IDENTIFIER EQ statement

;

expression : expression ADD term | expression SUB term

;

term : term MUL factor | term DIV factor | factor

;

factor : OPEN\_ROUND\_BRACKET expression CLOSED\_ROUND\_BRACKET | INTEGER | IDENTIFIER | IDENTIFIER OPEN\_RIGHT\_BRACKET INTEGER CLOSED\_RIGHT\_BRACKET

;

ioStatement : READ IDENTIFIER | WRITE IDENTIFIER | WRITE CONSTANT

;

condition : expression relation expression

;

relation : LT | LE | EQ | NE | GT | GE

%%

yyerror(char \*s)

{

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

}

extern FILE \*yyin;

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

}

p1.txt

start {

integer a;

integer b;

integer c;

integer max;

read a;

read b;

read c;

if(a>b&&a>c){

max=a;

}

else{

if(b>c&&b>a){

max=b;

}

else{

max=c;

}

}

write max;

}