**Objective**

To develop a compiler for the ring language

**Grammar**

Program —> {statement}

Statement —> ‘package’ <Identifier> { ‘.’ <Identifier> } [‘{‘ {statement} ‘}’] [‘end’|’endpackage’]

Statement —> ‘class’ <Identifier> [ ‘from’|’:’|’<’ <Identifier> ] [‘{‘ {statement} ‘}’][‘end’|’endclass’]

Statement —> ‘func’|’def’ <Identifier> [ParaList] [‘{‘ {statement} ‘}’][‘end’|’endfunc’]

Statement —> ‘import’ <Identifier> { ‘.’ <Identifier> }

Statement —> ‘private’

Statement —> ‘load’ [‘package’] <Literal>

Statement —> ‘loadsyntax’ <Literal>

Statement —> ‘changeringkeyword’ <OldKeyword> <NewKeyword>

Statement —> ‘changeringoperator’ <OldOperator> <NewOperator>

Statement —> ‘see’|’put’ <Expr>

Statement —> ‘give’|’get’ <Identifier>

Statement —> ‘if’ <Expr> [‘{‘] {statement} [ {‘but’|’elseif’ <Expr> {Statement} } ] [‘else’ {Statement} ] ‘ok’|’end’|’}’

Statement —> ‘Switch’ <Expr> [‘{‘] { ‘on’|’case’ <Expr> {statement} } [‘other’ {Statement} ] ‘off’|’end’|’}’

Statement —> ‘for’ <Identifier> ‘=’ <Expr> ‘to’ <Expr> [ ‘step’ <Expr> ] [‘{‘] {Statement} ‘next’|’end’|’}’

Statement —> ‘for’ <Identifier> ‘in’ <Expr> [ ‘step’ <Expr> ] [‘{‘] {statement} ‘next’|’end’|’}’

Statement —> ‘while’ <Expr> [‘{‘] {statement} ‘end’|’}’

Statement —> ‘do’ {statement} ‘again’ <Expr>

Statement —> ‘try’ {statement} [‘{‘] ‘catch’ {statement} ‘done’|’end’|’}’

Statement —> ‘return’ <Expr>

Statement —> ‘bye’

Statement —> ‘exit’

Statement —> ‘loop’

Statement —> <Expr>

Statement —> epslion

ParaList —> epslion

ParaList —> [‘(‘] <Identifier> [{ ‘,’ <Identifier> }] [‘)’]

**Implementation Language**

Lexical analyzer and parser generating tools: flex and bison

C Language

C18 Standard

**Type of Parser**

Bottom-up parser

**Methodology**

This parser uses bottom-up parsing to parse the given grammar as aforementioned. It consists of two components: a lexer which scans tokens from the input file and returns them to the parser which then uses the values from those tokens to parse the appropriae grammar productions. If a token is expected and it isn’t found, then it prints the error occurred with tthe line number at which the error occurred and the correct token that was expected.

**User Documentation/Readme**

To run the parser use the following commands:

1. bison -d ring.y

2. flex ring.l

3. gcc lex.yy.c ring.tab.c -o output

The first command causes bison to define the necessary files, the 2nd command processes the lexer and the third command compiles the resultant lexer C file produced by lex with the ring.tab.c file produced by Bison.

**Code**

**ring.y**

%{

#include<stdio.h>

#include<stdlib.h>

int yylex();

int yyerror();

extern FILE \*yyin;

extern char arr[100];

extern int i;

%}

%union {

char \*str;

}

%token <str>IF

%token <str>ELSE

%token <str>ELSEIF

%token <str>FOR

%token <str>TO

%token <str>STEP

%token <str>FUNC

%token <str>ENDFUNC

%token <str>WHILE

%token <str>END

%token <str>DO

%token <str>AGAIN

%token <str>TRY

%token <str>CATCH

%token <str>SWITCH

%token <str>CASE

%token <str>OTHER

%token <str>PACKAGE

%token <str>DOT

%token <str>ENDPACKAGE

%token <str>SEE

%token <str>LITERAL

%token <str>GIVE

%token <str>CLASS

%token <str>FROM

%token <str>ENDCLASS

%token <str>LOAD

%token <str>IDENTIFIER

%token <str>OPERATOR

%token <str>LBR

%token <str>RBR

%token <str>COMMA

%token <str>LR

%token <str>RR

%token <str>EOP

%define parse.error verbose

%%

stmt:program EOP

;

program:Statement

;

Statement:if\_statement

|for\_loop

|function\_statement

|while\_loop

|do\_again\_loop

|try\_catch\_statement

|switch\_statement

|package\_statement

|see\_statement

|give\_statement

|class\_statement

|load\_statement

|expr

;

if\_statement:IF expr lbr\_d stmt\_prime\_4 elseif\_expr\_stmt\_p else\_STMT\_d RBR Statement

;

for\_loop:FOR expr TO expr step\_expr\_d lbr\_d2 STMT\_prime\_3 RBR Statement

;

while\_loop:WHILE expr lbr\_d4 stmt\_prime\_7 RBR Statement

;

do\_again\_loop:DO stmt\_prime\_8 AGAIN expr Statement

;

try\_catch\_statement:TRY stmt\_prime\_9 lbr\_d5 CATCH stmt\_prime\_10 RBR Statement

;

switch\_statement:SWITCH expr lbr\_d1 case\_expr\_statement\_p other\_STMT\_d RBR Statement

;

package\_statement:PACKAGE IDENTIFIER dot\_id\_p lbr\_stmt\_rbr\_d ENDPACKAGE Statement

;

see\_statement:SEE expr Statement

|SEE LITERAL Statement

;

give\_statement:GIVE expr Statement

|GIVE LITERAL Statement

;

class\_statement:CLASS IDENTIFIER from\_colon\_lt\_id\_prime lbr\_stmt\_rbr\_d1 ENDCLASS Statement

;

load\_statement:LOAD PACKAGE LITERAL

;

from\_colon\_lt\_id\_prime:FROM IDENTIFIER

|

;

lbr\_stmt\_rbr\_d1: LBR stmt\_prime\_2 RBR

|

;

stmt\_prime\_2:stmt\_prime\_2 IDENTIFIER

|stmt\_prime\_2 expr

|

;

dot\_id\_p:DOT IDENTIFIER dot\_id\_p

|

;

lbr\_stmt\_rbr\_d:LBR stmt\_prime\_1 RBR

|

;

stmt\_prime\_1:stmt\_prime\_1 IDENTIFIER

|stmt\_prime\_1 expr

|

;

lbr\_d1:LBR

|

;

case\_expr\_statement\_p:case\_expr\_statement\_p CASE expr stmt\_prime\_5

|

;

other\_STMT\_d:OTHER STMT\_prime\_2

|

;

STMT\_prime\_2:STMT\_prime\_2 Statement

|

;

stmt\_prime\_5:stmt\_prime\_5 IDENTIFIER

|stmt\_prime\_5 expr

|

;

stmt\_prime\_9:stmt\_prime\_9 IDENTIFIER

|stmt\_prime\_9 expr

|

;

lbr\_d5:LBR

|

;

stmt\_prime\_10:stmt\_prime\_10 IDENTIFIER

|stmt\_prime\_10 expr

|

;

stmt\_prime\_8:stmt\_prime\_8 IDENTIFIER

|stmt\_prime\_8 expr

|

;

lbr\_d4:LBR

|

;

stmt\_prime\_7:stmt\_prime\_7 IDENTIFIER

|stmt\_prime\_7 expr

|

;

function\_statement:FUNC IDENTIFIER paralist\_d lbr\_stmt\_rbr\_d2 ENDFUNC Statement

;

paralist\_d:lr\_d IDENTIFIER COMMA IDENTIFIER rr\_d

|

;

lr\_d:LR

|

;

rr\_d:RR

|

;

lbr\_stmt\_rbr\_d2:LBR stmt\_prime\_3 RBR

|

;

stmt\_prime\_3:stmt\_prime\_3 IDENTIFIER

|stmt\_prime\_3 expr

|

;

step\_expr\_d:STEP expr

|

;

lbr\_d2:LBR

|

;

STMT\_prime\_3:STMT\_prime\_3 Statement

|

;

lbr\_d:LBR

|

;

stmt\_prime\_4:stmt\_prime\_4 IDENTIFIER

|stmt\_prime\_4 expr

|

;

elseif\_expr\_stmt\_p:elseif\_expr\_stmt\_p ELSEIF expr STMT\_p

|

;

STMT\_p:STMT\_p Statement

|

;

else\_STMT\_d:ELSE STMT\_prime\_1

|

;

STMT\_prime\_1:STMT\_prime\_1 Statement

|

;

expr:IDENTIFIER OPERATOR IDENTIFIER

;

%%

#ifdef YYDEBUG

yydebug = 1;

#endif

void main()

{

//printf("\n Enter the expression:\n");

yyin=fopen("test.txt","r");

do{

if(yyparse())

{

printf("\nFailure:\n");

exit(0);

}

}while(!feof(yyin));

printf("\nSuccess\n");

return 0;}

**ring.l**

%{

#include "ring.tab.h"

char arr[100];

int i = 0;

int flag = 0;

%}

%%

"if" {yylval.str = strdup(yytext);return IF;}

"else" {yylval.str = strdup(yytext);return ELSE;}

"elseif" {yylval.str = strdup(yytext);return ELSEIF;}

"+" {yylval.str = strdup(yytext);return OPERATOR;}

"-" {yylval.str = strdup(yytext);return OPERATOR;}

"\*" {yylval.str = strdup(yytext);return OPERATOR;}

"/" {yylval.str = strdup(yytext);return OPERATOR;}

"%" {yylval.str = strdup(yytext);return OPERATOR;}

"<" {yylval.str = strdup(yytext);return OPERATOR;}

"<=" {yylval.str = strdup(yytext);return OPERATOR;}

">" {yylval.str = strdup(yytext);return OPERATOR;}

">=" {yylval.str = strdup(yytext);return OPERATOR;}

"=" {yylval.str = strdup(yytext);return OPERATOR;}

"|" {yylval.str = strdup(yytext);return OPERATOR;}

"^" {yylval.str = strdup(yytext);return OPERATOR;}

"&" {yylval.str = strdup(yytext);return OPERATOR;}

"and" {yylval.str = strdup(yytext);return OPERATOR;}

"or" {yylval.str = strdup(yytext);return OPERATOR;}

"not" {yylval.str = strdup(yytext);return OPERATOR;}

"for" {yylval.str = strdup(yytext);return FOR;}

"to" {yylval.str = strdup(yytext);return TO;}

"step" {yylval.str = strdup(yytext);return STEP;}

"func" {yylval.str = strdup(yytext);return FUNC;}

"endfunc" {yylval.str = strdup(yytext);return ENDFUNC;}

"while" {yylval.str = strdup(yytext);return WHILE;}

"end" {yylval.str = strdup(yytext);return END;}

"do" {yylval.str = strdup(yytext);return DO;}

"again" {yylval.str = strdup(yytext);return AGAIN;}

"try" {yylval.str = strdup(yytext);return TRY;}

"catch" {yylval.str = strdup(yytext);return CATCH;}

"switch" {yylval.str = strdup(yytext);return SWITCH;}

"case" {yylval.str = strdup(yytext);return CASE;}

"other" {yylval.str = strdup(yytext);return OTHER;}

"package" {yylval.str = strdup(yytext);return PACKAGE;}

"." {yylval.str = strdup(yytext);return DOT;}

"endpackage" {yylval.str = strdup(yytext);return ENDPACKAGE;}

"see" {yylval.str = strdup(yytext);return SEE;}

"give" {yylval.str = strdup(yytext);return GIVE;}

"class" {yylval.str = strdup(yytext);return CLASS;}

"from" {yylval.str = strdup(yytext);return FROM;}

"endclass" {yylval.str = strdup(yytext);return ENDCLASS;}

"load" {yylval.str = strdup(yytext);return LOAD;}

"\""([^\n\"\\]\*(\\[.\n])\*)\*"\"" {yylval.str = strdup(yytext);return LITERAL;}

"{" {yylval.str = strdup(yytext);return LBR;}

"}" {yylval.str = strdup(yytext);return RBR;}

"(" {yylval.str = strdup(yytext);return LR;}

")" {yylval.str = strdup(yytext);return RR;}

"," {yylval.str = strdup(yytext);return COMMA;}

"@" {yylval.str = strdup(yytext);return EOP;}

[aA-zZ\_0-9]+ {yylval.str = strdup(yytext);return IDENTIFIER;}

%%

void yyerror(char \*msg) {

printf("error\n");

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

printf("line number: %d, msg: %s, char: %s\n", yylineno, msg, yytext );

}

int yywrap()

{

return 1;

}

**Snapshot of Sample Input and Output**

