CD LAB MINI PROJECT

Keya Sukthankar 160905186

Hazlin Dsouza 160905013

Dia Majumder 160905100

Semester VI, CSE B

**OBJECTIVE**: To design a lexical analyser and parser for a subset of MATLAB language.

**GRAMMAR**:

translation\_unit: statlist

| ‘function’ func\_decl eostmt statlist

;

Statlist: stat

| statlist stat

;

stat: globalstat

| clearstat

| assgstat

| expnstat

| selection

| iteration

| jump

;

globalstat: ‘global’ idlist eostmt

;

idlist: ID

| idlist ID

;

eostmt: ‘ , ’

| ‘ ; ‘

| ‘\n’

;

clearstat: ‘clear’ idlist eostmt

;

assgstat: assg\_exp eostmt

;

assg\_exp: postfix ‘ = ’ expn

;

postfix: primary\_exp

| array\_exp

| postfix ‘transpose’

| postfix ‘ ./’ ‘

;

expn: log\_exp

| expn ‘ : ’ log\_exp

;

log\_exp: equ\_exp

| log\_exp LOGOP equ\_exp

;

equ\_exp: rel\_exp

| equ\_exp EQUOP rel\_exp

;

rel\_exp: add\_exp

| rel\_exp RELOP add\_exp

;

add\_exp: mul\_exp

| add\_exp ADDOP mul\_exp

;

mul\_exp: unary\_exp

| mul\_exp MULOP unary\_exp

;

unary\_exp: postfix

| ‘ ~ ’ postfix

;

primary\_exp: ID

| NUM

| ‘ ( ‘ expn ‘ ) ‘

| ‘ [ ‘ ‘ ] ‘

| ‘ [ ‘ arraylist ‘ ] ‘

;

arraylist: arrayelement

| arraylist arrayelement

;

arrayelement: expn

| expnstat

;

expnstat: eostmt

| expn eostmt

;

array\_exp: ID ‘ ( ‘ index\_exp\_list ‘ ) ‘

;

index\_exp\_list: index\_exp

| index\_exp\_list ‘ , ‘ index\_exp

;

index\_exp: ‘ : ‘

| expn

;

selection: ‘if’ expn statlist ‘end’ eostmt

| ‘if’ expn statlist ‘else’ statlist ‘end’ eostmt

| ‘if’ expn statlist elseif ‘end’ eostmt

| ‘if’ expn statlist elseif ‘else’ statlist ‘end eostmt

;

elseif: ‘elseif’ expn statlist

| elseif ‘elseif’ expn statlist

;

iteration: ‘while’ expn statlist ‘end’

| ‘for’ ID ‘ = ’ expn statlist ‘end’ eostmt

| ‘for’ ‘ ( ‘ ID ‘ = ‘ expn ‘ ) ‘ statlist ‘end’ eostmt

;

jump: ‘break’ eostmt

| ‘return’ eostmt

;

func\_decl: func\_decl\_lhs

| func\_return\_list ‘ = ‘ func\_decl\_lhs

;

func\_decl\_lhs: ID

| ID ‘ ( ‘ ‘ ) ‘

| ID ‘ ( ‘ func\_ident\_list ‘ ) ‘

;

func\_return\_list: ID

| ‘ [ ‘ func\_ident\_list ‘ ] ‘

;

func\_ident\_list: ID

| func\_ident\_list ‘ , ‘ ID

;

**LANGUAGE USED FOR IMPLEMENTATION**: Flex, Bison and C. (C only for symbol table.)

**TYPE OF PARSER**: Bottom up.

**METHOD**:

This project takes a matlab script as input, parses it and tells whether it is a valid script or not. The lexical analyser is implemented by the flex file. It searches for tokens matching the rules specified, in the input. When it finds a match, it sends the generated token to the parser, which is implemented using the bison file. The parser searches for a production to match the token. If the parser is able to match all the tokens of the input file to the given grammar productions, it is successfully parsed.

**USER DOCUMENTATION**:

bison file: parser.y: contains context-free-grammar for MATLAB.

flex file: lex.l: generate tokens for the given grammar

c file: symtab.c: contains functions for searching and inserting tokens in symbol table. (only identfiers, numbers and operators are stored.)

VARIABLES AND FUNCTIONS:

translation\_unit It is start symbol of grammar

statlist list of statements

eostmt end of statement

func\_decl function declaration

stat statement

assgstat assignment statement.

Expnstat expression statement

Selection If else statement

Iteration for and while loops

Jump unconditional jump

Idlist list of Identifiers.

assg\_exp assignment expression.

Postfix postfix expression.

unary\_exp single operand expression.

Log\_exp logical expression

Equ\_exp equality/ inequality expression

Rel\_exp relational expression

Add\_exp addition expression

Mul\_exp multiplication expression

NL \n

OB parenthesis open

CB parenthesis close

SQ square bracket open

SCB square bracket close

NCTRANSPOSE ./’ operator

COL :

COM ,

NOT ~

MULOP \*, /, ^, .\*, ./, .^

ADDOP +, -

RELOP <, >, <=, >=

EQUOP ==, ~=

LOGOP &, |

EQ =

SEMICOL ;

NUM numbers

ID identifiers

keywords: TRANSPOSE, GLOBAL, CLEAR, IF, ELSE, ELSEIF, END, FOR, WHILE, BREAK, RETURN, FUNCTION

Token structure which contains name and type of token

List structure to create linked list of tokens

Table array of linked lists, actually stores the symbol table

void init() initialize table to NULL

int hash(char \*) hash function

int search(char \*) search for token in symbol table

void insert(token) insert a token in symbol table

CODE:

*symtab.c*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define len 100

typedef struct token{

char name[100];

char type[50];

}token;

struct list{

token tok;

struct list \*next;

};

struct list \*table[len];

void init()

{

for(int i=0;i<len;i++)

table[i]=NULL;

}

int hash(char str[])

{

int x=0;

for (int i = 0; str[i] != '\0'; i++)

x+=str[i];

x = x%len;

return x;

}

int search(char str[])

{

int x = hash(str);

struct list \*p=table[x];

while(p!=NULL)

{ if(strcmp(p->tok.name,str)==0)

break;

p=p->next;

}

if(p==NULL) return 0;

return 1;

}

void insert(token tk)

{

if(search(tk.name))

return;

int x=hash(tk.name);

struct list \*newptr=(struct list \*)malloc(sizeof(struct list));

newptr->tok=tk;

//newptr->x=x;

newptr->next=NULL;

if(table[x]==NULL)

table[x]=newptr;

else

{

struct list \*p=table[x];

while(p->next!=NULL)

p=p->next;

p->next=newptr;

}

}

void print()

{

int k=0;

for(int i=0;i<len;i++)

{

struct list \*p=table[i];

while(p!=NULL)

{

token t=p->tok;

printf("%d\t%s\t%s\n",k,t.name,t.type);

k++;

p=p->next;

}

}

}

*lex.l*

%{

#include "symtab.c"

#include "parser.tab.h"

int j=0;

token t[len];

%}

%%

"%".\*"\n" {;}

\n { printf("\n"); return NL;}

\t {;}

"(" { printf("%s ",yytext); return OB;}

")" { printf("%s ",yytext); return CB;}

"[" { printf("%s ",yytext); return SB;}

"]" { printf("%s ",yytext); return SCB;}

"transpose" { printf("%s ",yytext); return TRANSPOSE;}

".\'" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op");

insert(t[j]); j++; return NCTRANSPOSE;}

":" { printf("%s ",yytext); return COL;}

"," { printf("%s ",yytext); return COM;}

"~" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op"); insert(t[j]); j++; return NOT;}

"\*"|"/"|"^"|".\*"|"./"|".^" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op");

insert(t[j]); j++; return MULOP;}

"+"|"-" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op"); insert(t[j]); j++; return ADDOP;}

"<"|">"|"<="|">=" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op");

insert(t[j]); j++; return RELOP;}

"=="|"~=" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op"); insert(t[j]); j++; return EQUOP;}

"&"|"|" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op"); insert(t[j]); j++; return LOGOP;}

"=" { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"op"); insert(t[j]); j++; return EQ;}

";" { printf("%s ",yytext); return SEMICOL;}

"global" { printf("%s ",yytext); return GLOBAL;}

"clear" { printf("%s ",yytext); return CLEAR;}

"if" { printf("%s ",yytext); return IF;}

"else" { printf("%s ",yytext); return ELSE;}

"elseif" { printf("%s ",yytext); return ELSEIF;}

"end" { printf("%s ",yytext); return END;}

"for" { printf("%s ",yytext); return FOR;}

"while" { printf("%s ",yytext); return WHILE;}

"break" { printf("%s ",yytext); return BREAK;}

"return" { printf("%s ",yytext); return RETURN;}

"function" { printf("%s ",yytext); return FUNCTION;}

[0-9]+ {printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"num"); insert(t[j]); j++; return NUM;}

[a-z][a-zA-z\_0-9]\* { printf("%s ",yytext); strcpy(t[j].name,yytext); strcpy(t[j].type,"id");

insert(t[j]), j++; return ID;}

. { printf("%s ",yytext);}

%%

int yywrap()

{

return 1;

}

*parser.y*

%{

#include <stdio.h>

#include <stdlib.h>

int yylex();

int yyerror();

extern FILE \*yyin;

%}

%token ID NUM OB CB SB SCB TRANSPOSE NCTRANSPOSE COL COM NOT MULOP ADDOP RELOP EQUOP LOGOP EQ SEMICOL NL GLOBAL CLEAR IF ELSE ELSEIF END FOR WHILE BREAK RETURN FUNCTION

%%

translation\_unit: statlist

| FUNCTION func\_decl eostmt statlist

;

statlist: stat

| statlist stat

;

stat: globalstat

| clearstat

| assgstat

| expnstat

| selection

| iteration

| jump

;

globalstat: GLOBAL idlist eostmt

;

idlist: ID

| idlist ID

;

eostmt: COM

| SEMICOL

| NL

;

clearstat: CLEAR idlist eostmt

;

assgstat: assg\_exp eostmt

;

assg\_exp: postfix EQ expn

;

postfix: primary\_exp

| array\_exp

| postfix TRANSPOSE

| postfix NCTRANSPOSE

;

expn: log\_exp

| expn COL log\_exp

;

log\_exp: equ\_exp

| log\_exp LOGOP equ\_exp

;

equ\_exp: rel\_exp

| equ\_exp EQUOP rel\_exp

;

rel\_exp: add\_exp

| rel\_exp RELOP add\_exp

;

add\_exp: mul\_exp

| add\_exp ADDOP mul\_exp

;

mul\_exp: unary\_exp

| mul\_exp MULOP unary\_exp

;

unary\_exp: postfix

| NOT postfix

;

primary\_exp: ID

| NUM

| OB expn CB

| SB SCB

| SB arraylist SCB

;

arraylist: arrayelement

| arraylist arrayelement

;

arrayelement: expn

| expnstat

;

expnstat: eostmt

| expn eostmt

;

array\_exp: ID OB index\_exp\_list CB

;

index\_exp\_list: index\_exp

| index\_exp\_list COM index\_exp

;

index\_exp: COL

| expn

;

selection: IF expn statlist END eostmt

| IF expn statlist ELSE statlist END eostmt

| IF expn statlist elseif END eostmt

| IF expn statlist elseif ELSE statlist END eostmt

;

elseif: ELSEIF expn statlist

| elseif ELSEIF expn statlist

;

iteration: WHILE expn statlist END

| FOR ID EQ expn statlist END eostmt

| FOR OB ID EQ expn CB statlist END eostmt

;

jump: BREAK eostmt

| RETURN eostmt

;

func\_decl: func\_decl\_lhs

| func\_return\_list EQ func\_decl\_lhs

;

func\_decl\_lhs: ID

| ID OB CB

| ID OB func\_ident\_list CB

;

func\_return\_list: ID

| SB func\_ident\_list SCB

;

func\_ident\_list: ID

| func\_ident\_list COM ID

;

%%

int yyerror()

{

printf("failed\n");

return 1;

}

void main()

{

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

do{

if(yyparse())

{

printf("failure\n");

exit(1);

}

}while(!feof(yyin));

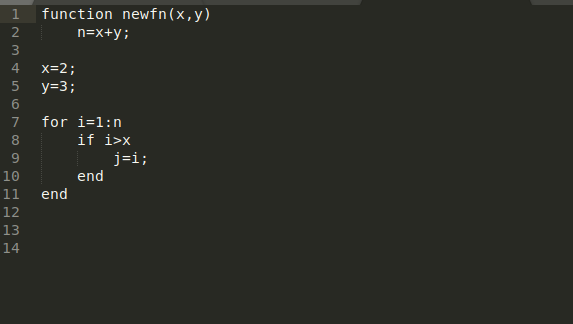
printf("success\n");

print();

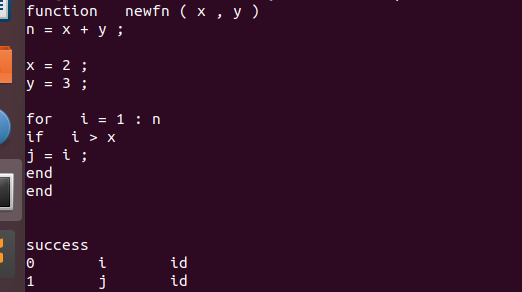
}

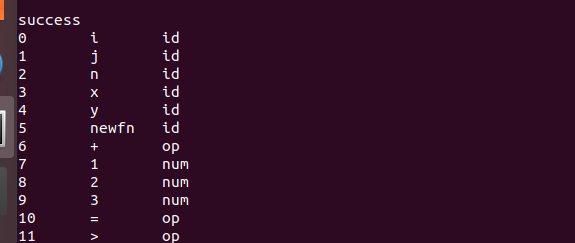
SAMPLE INPUT AND OUTPUT:

**Input:**

****

**Output:**

****

****