DEPARTMENT OF INFORMATION TECHNOLOGY



PRACTICLE FILE

COMPILER DESIGN

(160601)

SUBMITTED TO: SUBMITTED BY:

ABHILASH SONKER DHEERAJ KUSHWAH

Asst. Prof. 0901IT181023

NAMRATA AGARWAL Session JANUARY-APRIL, 2021

Asst. Prof. IT 3rd YEAR (6th Sem.)

Content

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Experiment name | Date of Practical | Signature of Teacher |
| 1 | Write a program to convert NFA to DFA. |  |  |
| 2 | Write a program to minimize DFA. |  |  |
| 3 | Develop a lexical analyzer to recognize a few patterns |  |  |
| 4 | Write a program to parse using Brute force technique of Top down parsing. |  |  |
| 5 | Develop LL (1) parser (Construct parse table also). |  |  |
| 6 | Develop an operator precedence parser (Construct parse table also). |  |  |
| 7 | Develop a recursive descent parser. |  |  |
| 8 | Write a program for generating for various intermediate code forms.  i) Three address code  ii) Polish notation |  |  |
| 9 | Write a program to simulate Heap storage allocation strategy. |  |  |
| 10 | Generate Lexical analyzer using LEX. |  |  |
| 11 | Write a program to calculate first and follow of given grammer. |  |  |
| 12 | Given any intermediate code form implement code optimization techniques. |  |  |

**Aim: Write a program to convert NFA to DFA.**

Program:

#include <cstdio>

#include <fstream>

#include <iostream>

#include <bitset>

#include <vector>

#include <cstring>

#include <cstdlib>

#include <algorithm>

#include <queue>

#include <set>

#define MAX\_NFA\_STATES 10

#define MAX\_ALPHABET\_SIZE 10

using namespace std;

// Representation of an NFA state

class NFAstate

{

public:

int transitions[MAX\_ALPHABET\_SIZE][MAX\_NFA\_STATES];

NFAstate()

{

for (int i = 0; i < MAX\_ALPHABET\_SIZE; i++)

for (int j = 0; j < MAX\_NFA\_STATES; j++)

transitions[i][j] = -1;

}

}\*NFAstates;

// Representation of a DFA state

struct DFAstate

{

bool finalState;

bitset<MAX\_NFA\_STATES> constituentNFAstates;

bitset<MAX\_NFA\_STATES> transitions[MAX\_ALPHABET\_SIZE];

int symbolicTransitions[MAX\_ALPHABET\_SIZE];

};

set<int> NFA\_finalStates;

vector<int> DFA\_finalStates;

vector<DFAstate\*> DFAstates;

queue<int> incompleteDFAstates;

int N, M; // N -> No. of stattes, M -> Size of input alphabet

// finds the epsilon closure of the NFA state "state" and stores it into "closure"

void epsilonClosure(int state, bitset<MAX\_NFA\_STATES> &closure)

{

for (int i = 0; i < N && NFAstates[state].transitions[0][i] != -1; i++)

if (closure[NFAstates[state].transitions[0][i]] == 0)

{

closure[NFAstates[state].transitions[0][i]] = 1;

epsilonClosure(NFAstates[state].transitions[0][i], closure);

}

}

// finds the epsilon closure of a set of NFA states "state" and stores it into "closure"

void epsilonClosure(bitset<MAX\_NFA\_STATES> state,

bitset<MAX\_NFA\_STATES> &closure)

{

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

if (state[i] == 1)

epsilonClosure(i, closure);

}

// returns a bitset representing the set of states the NFA could be in after moving

// from state X on input symbol A

void NFAmove(int X, int A, bitset<MAX\_NFA\_STATES> &Y)

{

for (int i = 0; i < N && NFAstates[X].transitions[A][i] != -1; i++)

Y[NFAstates[X].transitions[A][i]] = 1;

}

// returns a bitset representing the set of states the NFA could be in after moving

// from the set of states X on input symbol A

void NFAmove(bitset<MAX\_NFA\_STATES> X, int A, bitset<MAX\_NFA\_STATES> &Y)

{

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

if (X[i] == 1)

NFAmove(i, A, Y);

}

int main()

{

int i, j, X, Y, A, T, F, D;

// read in the underlying NFA

ifstream fin("NFA.txt");

fin >> N >> M;

NFAstates = new NFAstate[N];

fin >> F;

for (i = 0; i < F; i++)

{

fin >> X;

NFA\_finalStates.insert(X);

}

fin >> T;

while (T--)

{

fin >> X >> A >> Y;

for (i = 0; i < Y; i++)

{

fin >> j;

NFAstates[X].transitions[A][i] = j;

}

}

fin.close();

// construct the corresponding DFA

D = 1;

DFAstates.push\_back(new DFAstate);

DFAstates[0]->constituentNFAstates[0] = 1;

epsilonClosure(0, DFAstates[0]->constituentNFAstates);

for (j = 0; j < N; j++)

if (DFAstates[0]->constituentNFAstates[j] == 1 && NFA\_finalStates.find(

j) != NFA\_finalStates.end())

{

DFAstates[0]->finalState = true;

DFA\_finalStates.push\_back(0);

break;

}

incompleteDFAstates.push(0);

while (!incompleteDFAstates.empty())

{

X = incompleteDFAstates.front();

incompleteDFAstates.pop();

for (i = 1; i <= M; i++)

{

NFAmove(DFAstates[X]->constituentNFAstates, i,

DFAstates[X]->transitions[i]);

epsilonClosure(DFAstates[X]->transitions[i],

DFAstates[X]->transitions[i]);

for (j = 0; j < D; j++)

if (DFAstates[X]->transitions[i]

== DFAstates[j]->constituentNFAstates)

{

DFAstates[X]->symbolicTransitions[i] = j;

break;

}

if (j == D)

{

DFAstates[X]->symbolicTransitions[i] = D;

DFAstates.push\_back(new DFAstate);

DFAstates[D]->constituentNFAstates

= DFAstates[X]->transitions[i];

for (j = 0; j < N; j++)

if (DFAstates[D]->constituentNFAstates[j] == 1

&& NFA\_finalStates.find(j) != NFA\_finalStates.end())

{

DFAstates[D]->finalState = true;

DFA\_finalStates.push\_back(D);

break;

}

incompleteDFAstates.push(D);

D++;

}

}

}

// write out the corresponding DFA

ofstream fout("DFA.txt");

fout << D << " " << M << "\n" << DFA\_finalStates.size();

for (vector<int>::iterator it = DFA\_finalStates.begin(); it

!= DFA\_finalStates.end(); it++)

fout << " " << \*it;

fout << "\n";

for (i = 0; i < D; i++)

{

for (j = 1; j <= M; j++)

fout << i << " " << j << " "

<< DFAstates[i]->symbolicTransitions[j] << "\n";

}

fout.close();

return 0;

}

Output:

Input file

NFA.txt

4 2

2 0 1

4

0 1 2 1 2

1 1 2 1 2

2 2 2 1 3

3 1 2 1 2

Output file

4 2

3 0 1 3

0 1 1

0 2 2

1 1 1

1 2 3

2 1 2

2 2 2

3 1 1

3 2 2

**Aim: Write a program to minimize DFA.**

Program:

#include<stdio.h>           
  
#include<string.h>  
  
#define STATES 50  
  
struct Dstate  
  
{  
  
char name;  
  
char StateString[STATES+1];  
  
char trans[10];      
  
int is\_final;  
  
}Dstates[50];  
  
struct tran  
  
{  
  
char sym;  
  
int tostates[50];  
  
int notran;  
  
};  
  
struct state  
  
{  
  
int no;  
  
struct tran tranlist[50];  
  
};  
  
int stackA[100],stackB[100],c[100],Cptr=-1,Aptr=-1,Bptr=-1;  
  
struct state States[10];  
  
char temp[STATES+1],inp[10];  
  
int nos,noi,nof,j,k,nods=-1;  
  
void pushA(int z)  
  
{  
  
stackA[++Aptr]=z;  
  
}  
  
void pushB(int z)  
  
{  
  
stackB[++Bptr]=z;  
  
}  
  
  
  
  
int popA()  
  
{  
  
return stackA[Aptr--];  
  
}  
  
void copy(int i)  
  
{  
  
char temp[STATES+1]=" ";  
  
int k=0;  
  
Bptr=-1;  
  
strcpy(temp,Dstates[i].StateString);  
  
while(temp[k]!='\0')  
  
{  
  
pushB(temp[k]-'0');  
  
k++;  
  
}  
  
}  
  
int popB()  
  
{  
  
return stackB[Bptr--];  
  
}  
  
int peekA()  
  
{  
  
return stackA[Aptr];  
  
}  
  
int peekB()  
  
{  
  
return stackA[Bptr];  
  
}  
  
int seek(int arr[],int ptr,int s)  
  
{  
  
int i;  
  
for(i=0;i<=ptr;i++)  
  
{  
  
  
  
  
if(s==arr[i])  
  
return 1;  
  
}  
  
return 0;  
  
}  
  
void sort()  
  
{  
  
int i,j,temp;  
  
for(i=0;i<Bptr;i++)  
  
{  
  
for(j=0;j<(Bptr-i);j++)  
  
{  
  
if(stackB[j]>stackB[j+1])  
  
{  
  
temp=stackB[j];     
  
stackB[j]=stackB[j+1];  
  
stackB[j+1]=temp;  
  
}  
  
}  
  
  }  
  
    }  
  
void tostring()  
  
{  
  
int i=0;  
  
sort();          
  
for(i=0;i<=Bptr;i++)  
  
{        
  
temp[i]=stackB[i]+'0';  
  
}  
  
temp[i]='\0';  
  
}  
  
void display\_DTran()  
  
{  
  
int i,j;  
  
printf("\n\t\t DFA transition table");  
  
printf("\n\t\t ---------------------------------------------- ");  
  
printf("\n States \tString \tInputs\n");  
  
for(i=0;i<noi;i++)  
  
{  
  
printf("\t %c",inp[i]);  
  
}  
  
printf("\n\t ------------------------------------------------- ");  
  
for(i=0;i<nods;i++)  
  
{  
  
if(Dstates[i].is\_final==0)  
  
printf("\n%c",Dstates[i].name);  
  
else  
  
printf("\n\*%c",Dstates[i].name);  
  
printf("\t%s",Dstates[i].StateString);  
  
for(j=0;j<noi;j++)  
  
{  
  
printf("\t%c",Dstates[i].trans[j]);  
  
}  
  
    }  
  
     printf("\n");  
  
}  
  
void move(int st,int j)  
  
{  
  
int ctr=0;  
  
while(ctr<States[st].tranlist[j].notran)  
  
{  
  
pushA(States[st].tranlist[j].tostates[ctr++]);  
  
}  
  
}  
  
void lambda\_closure(int st)  
  
{  
  
int ctr=0,in\_state=st,curst=st,chk;  
  
while(Aptr!=-1)  
  
{  
  
curst=popA();  
  
ctr=0;  
  
in\_state=curst;  
  
while(ctr<=States[curst].tranlist[noi].notran)  
  
{  
  
chk=seek(stackB,Bptr,in\_state);  
  
if(chk==0)  
  
pushB(in\_state);  
  
in\_state=States[curst].tranlist[noi].tostates[ctr++];  
  
chk=seek(stackA,Aptr,in\_state);  
  
if(chk==0 && ctr<=States[curst].tranlist[noi].notran)  
  
pushA(in\_state);  
  
}  
  
     }  
  
}  
  
void main()  
  
{  
  
int i,final[20],start,fin=0;  
  
char c,ans,st[20];  
  
printf("\n Enter no of states in NFA:");  
  
scanf("%d",&nos);  
  
for(i=0;i<nos;i++)  
  
{  
  
States[i].no=i;  
  
}  
  
printf("\n Enter the start states:");  
  
scanf("%d",&start);  
  
printf("Enter the no of final states:");  
  
scanf("%d",&nof);  
  
printf("Enter the final states:\n");  
  
for(i=0;i<nof;i++)  
  
scanf("%d",&final[i]);  
  
printf("\n Enter the no of input symbols:");  
  
scanf("%d",&noi);  
  
c=getchar();  
  
printf("Enter the input symbols:\n");  
  
for(i=0;i<noi;i++)  
  
{  
  
scanf("%c",&inp[i]);  
  
c=getchar();  
  
}  
  
//g1inp[i]='e';  
  
inp=[i]=’e’;  
  
printf("\n Enter the transitions:(-1 to stop)\n");  
  
for(i=0;i<nos;i++)  
  
{  
  
for(j=0;j<=noi;j++)  
  
{  
  
States[i].tranlist[j].sym=inp[j];  
  
k=0;  
  
ans='y';  
  
while(ans=='y')  
  
{  
  
printf("move(%d,%c);",i,inp[j]);  
  
scanf("%d",&States[i].tranlist[j].tostates[k++]);  
  
if((States[i].tranlist[j].tostates[k-1]==-1))  
  
{  
  
k--;  
  
ans='n';  
  
break;  
  
                      }  
  
               }  
  
States[i].tranlist[j].notran=k;  
  
               }  
  
}  
  
i=0;nods=0,fin=0;  
  
pushA(start);  
  
lambda\_closure(peekA());  
  
tostring();  
  
Dstates[nods].name='A';  
  
nods++;  
  
strcpy(Dstates[0].StateString,temp);  
  
while(i<nods)  
  
{  
  
for(j=0;j<noi;j++)  
  
{  
  
fin=0;  
  
copy(i);  
  
while(Bptr!=-1)  
  
{  
  
move(popB(),j);  
  
}  
  
  
  
  
while(Aptr!=-1)  
  
lambda\_closure(peekA());  
  
tostring();  
  
for(k=0;k<nods;k++)  
  
{  
  
if((strcmp(temp,Dstates[k].StateString)==0))  
  
{  
  
Dstates[i].trans[j]=Dstates[k].name;  
  
break;  
  
}  
  
}  
  
if(k==nods)  
  
{  
  
nods++;  
  
for(k=0;k<nof;k++)  
  
{  
  
fin=seek(stackB,Bptr,final[k]);  
  
if(fin==1)  
  
{  
  
Dstates[nods-1].is\_final=1;  
  
break;  
  
}  
  
    }  
  
strcpy(Dstates[nods-1].StateString,temp);  
  
Dstates[nods-1].name='A'+nods-1;  
  
Dstates[i].trans[j]=Dstates[nods-1].name;  
  
}  
  
}  
  
i++;  
  
}  
  
display\_DTran();  
  
}

Output:

Enter the no of input symbols:2  
Enter the input symbols:  
a ,b  
Enter the transitions:(-1 to stop)  
move(0,a);-1  
move(0,b);-1  
move(0,e);1  
move(0,e);7  
move(0,e);-1  
move(1,a);-1  
move(1,b);-1  
move( 1,e);2  
move(1,e);4  
move(1,e);-1  
move(2,a);3  
move(2,a);3  
move(2,a);-1  
move(2,b);-1  
move(2,e);-1  
move(3,a);-1  
move(3,b);-1  
move(3,e);6  
move(3,e);-1  
move(4,a);-1  
move(4,b);-1  
move(4,e);-1  
move(5,a);-1  
move(5,b);-1  
move(5,e);6  
move(5,e);1  
move(5,e);-1  
move(6,a);-1  
move(6,b);-1  
move(6,e);-1  
move(7,a);-1  
move(7,b);-1  
move(7,e);-1

DFA transition table  
States         String             Inputs  
                                      a       b          
                           -------------------------------------------------  
A                  01247              B       C  
B                     36              C       C  
C                                     C      C

**Aim: Develop a lexical analyzer to recognize a few patterns.**

Program:

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

#define SIZE 128

#define NONE -1

#define EOS ‘\0’

#define NUM 256

#define KEYWORD 257

#define PAREN 258

#define ID 259

#define ASSIGN 260

#define REL\_OP 261

#define DONE 262

#define MAX 999

char lexemes[MAX];

char buffer[SIZE];

int lastchar = -1;

int lastentry = 0;

int tokenval=NONE;

int lineno=1;

struct entry

{

char \*lexptr;

int token;

}symtable[100];

struct entry keywords[]={“if”,KEYWORD,”else”,KEYWORD,”for”,KEYWORD,

“int”,KEYWORD,”float”,KEYWORD,”double”,KEYWORD,”char”,KEYWORD,

“struct”,KEYWORD,”return”,KEYWORD,0,0};

void Error\_Message(char \*m)

{

fprint(stderr,”line %d: %s”,lineno,m);

exit(1);

}

int look\_up(char s[])

{

int k;

for(k=lastentry;k>0;k--)

if(strcmp(symtable[k].lexptr,s)==0)

return k;

return 0;

}int insert(chars[],int tok)

{

int len;

len=strlen(s);

if(lastentry+1>=MAX)

Error\_Message(“Symbol Table is Full”);

if(lastchar+len+1>=MAX)

Error\_Message(“Lexemes Array is Full”);

lastentry++;

symtable[lastentry].token=tok;

symtable[lastentry].lexptr=&lexemes[lastcher+1];

lastchar = lastchar + len + 1;

strcpy(smtable[lastentry].lexptr,s);

return lastentry;

}

void Initialize()

{

struct entry \*ptr;

for(ptr=keywords;ptr->token;ptr++)

insert(ptr->lexptr,ptr->token);

}

int lexer()

{

int t;

int val,i=0;

while(1)

{

t=getchar();

if(t == ’’ || t==’\t’);

else if(t==’\n’)

lineno++;

else if(t == ’(‘ || t == ‘)’)

return PAREN;

else if(t==‘<’ ||t==‘>’ ||t==‘<=’ ||t==‘>=’ ||t == ‘!=’)

return REL\_OP;

else if(t == ’=’)

return ASSIGN;

else if(isdigit(t))

{

ungetc(t,stdin);

scanf(“%d”,&tokenval);

return NUM;

}

else if(isalpha(t))

{while(isalnum(t))

{

buffer[i]=t;

t=getchar();

i++;

if(i>=SIZE)

Error\_Message(“compiler error”);

}

buffer[i]=EOS;

if(t!=EOF)

ungetc(t,stdin);

val=look\_up(buffer);

if(val==0)

val=insert(buffer,ID);

tokenval=val;

return symtable[val].token;

}

else if(t==EOF)

return DONE;

else

{

tokenval=NONE;

return t;

}

}

}

void main()

{

int lookahead;

char ans;

clrscr();

printf(“\n]t]t Program for Lexical Analysis \n”);

Initialize();

printf(“\n Enter the expression and put ; at the end”);

printf(“\n Press Ctrl + Z to terminate... \n”);

lookahead=lexer();

while(lookahead!=DONE)

{

if(lookahead==NUM)

printf(“\n Number: %d”,tokenval);

if(lookahead==’+’|| lookahead==’-’|| lookahead==’\*’||

lookahead==’/’)

printf(“\n Operator”);

if(lookahead==PAREN)

printf(“\n Parentesis”);

if(lookahead==ID)printf(“\n Identifier: %s“,

symtable[tokenval].lexptr);

if(lookahead==KEYWORD)

printf(“\n Keyword);

if(lookahead==ASSIGN)

printf(“\n Assignment Operator”);

if(lookahead==REL\_OP)

printf(“\n Relataional Operator”);

lookahead=lexer();

}

}

Output:

Program for Lexical Analysis

Enter the expression and put ; at the end

Press Ctrl + Z to terminate ...

2+3

Number: 2

Operator

Number: 3

if(a<b) a=a+b;

Keyword

Parenthesis

Identifier: a

Relational Operator

Identifier: b

Parenthesis

Identifier: a

Assigment Operator

Identifier: a

Operator

Identifier: b

^Z

**Aim: Write a program to parse using Brute force technique of Top down parsing.**

Program:

**Aim: Develop LL (1) parser (Construct parse table also).**

Program:

int table[100][TSIZE];

char terminal[TSIZE];

char nonterminal[26];

struct product {

char str[100];

int len;

}pro[20];

// no of productions in form A->ß

int no\_pro;

char first[26][TSIZE];

char follow[26][TSIZE];

// stores first of each production in form A->ß

char first\_rhs[100][TSIZE];

// check if the symbol is nonterminal

int isNT(char c) {

return c >= 'A' && c <= 'Z';

}

// reading data from the file

void readFromFile() {

FILE\* fptr;

fptr = fopen("text.txt", "r");

char buffer[255];

int i;

int j;

while (fgets(buffer, sizeof(buffer), fptr)) {

printf("%s", buffer);

j = 0;

nonterminal[buffer[0] - 'A'] = 1;

for (i = 0; i < strlen(buffer) - 1; ++i) {

if (buffer[i] == '|') {

++no\_pro;

pro[no\_pro - 1].str[j] = '\0';

pro[no\_pro - 1].len = j;

pro[no\_pro].str[0] = pro[no\_pro - 1].str[0];

pro[no\_pro].str[1] = pro[no\_pro - 1].str[1];

pro[no\_pro].str[2] = pro[no\_pro - 1].str[2];

j = 3;

}

else {

pro[no\_pro].str[j] = buffer[i];

++j;

if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {

terminal[buffer[i]] = 1;

}

}

}

pro[no\_pro].len = j;

++no\_pro;

}

}

void add\_FIRST\_A\_to\_FOLLOW\_B(char A, char B) {

int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];

}

}

void add\_FOLLOW\_A\_to\_FOLLOW\_B(char A, char B) {

int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];

}

}

void FOLLOW() {

int t = 0;

int i, j, k, x;

while (t++ < no\_pro) {

for (k = 0; k < 26; ++k) {

if (!nonterminal[k]) continue;

char nt = k + 'A';

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

if (nt == pro[i].str[j]) {

for (x = j + 1; x < pro[i].len; ++x) {

char sc = pro[i].str[x];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FOLLOW\_B(sc, nt);

if (first[sc - 'A']['^'])

continue;

}

else {

follow[nt - 'A'][sc] = 1;

}

break;

}

if (x == pro[i].len)

add\_FOLLOW\_A\_to\_FOLLOW\_B(pro[i].str[0], nt);

}

}

}

}

}

}

void add\_FIRST\_A\_to\_FIRST\_B(char A, char B) {

int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^') {

first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];

}

}

}

void FIRST() {

int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_B(sc, pro[i].str[0]);

if (first[sc - 'A']['^'])

continue;

}

else {

first[pro[i].str[0] - 'A'][sc] = 1;

}

break;

}

if (j == pro[i].len)

first[pro[i].str[0] - 'A']['^'] = 1;

}

++t;

}

}

void add\_FIRST\_A\_to\_FIRST\_RHS\_\_B(char A, int B) {

int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

first\_rhs[B][i] = first[A - 'A'][i] || first\_rhs[B][i];

}

}

// Calculates FIRST(ß) for each A->ß

void FIRST\_RHS() {

int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_RHS\_\_B(sc, i);

if (first[sc - 'A']['^'])

continue;

}

else {

first\_rhs[i][sc] = 1;

}

break;

}

if (j == pro[i].len)

first\_rhs[i]['^'] = 1;

}

++t;

}

}

int main() {

readFromFile();

follow[pro[0].str[0] - 'A']['$'] = 1;

FIRST();

FOLLOW();

FIRST\_RHS();

int i, j, k;

// display first of each variable

printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

char c = pro[i].str[0];

printf("FIRST OF %c: ", c);

for (j = 0; j < TSIZE; ++j) {

if (first[c - 'A'][j]) {

printf("%c ", j);

}

}

printf("\n");

}

}

// display follow of each variable

printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

char c = pro[i].str[0];

printf("FOLLOW OF %c: ", c);

for (j = 0; j < TSIZE; ++j) {

if (follow[c - 'A'][j]) {

printf("%c ", j);

}

}

printf("\n");

}

}

// display first of each variable ß

// in form A->ß

printf("\n");

for (i = 0; i < no\_pro; ++i) {

printf("FIRST OF %s: ", pro[i].str);

for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j]) {

printf("%c ", j);

}

}

printf("\n");

}

terminal['$'] = 1;

terminal['^'] = 0;

// printing parse table

printf("\n");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LL(1) PARSING TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t--------------------------------------------------------\n");

printf("%-10s", "");

for (i = 0; i < TSIZE; ++i) {

if (terminal[i]) printf("%-10c", i);

}

printf("\n");

int p = 0;

for (i = 0; i < no\_pro; ++i) {

if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))

p = p + 1;

for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j] && j != '^') {

table[p][j] = i + 1;

}

else if (first\_rhs[i]['^']) {

for (k = 0; k < TSIZE; ++k) {

if (follow[pro[i].str[0] - 'A'][k]) {

table[p][k] = i + 1;

}

}

}

}

}

k = 0;

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

printf("%-10c", pro[i].str[0]);

for (j = 0; j < TSIZE; ++j) {

if (table[k][j]) {

printf("%-10s", pro[table[k][j] - 1]);

}

else if (terminal[j]) {

printf("%-10s", "");

}

}

++k;

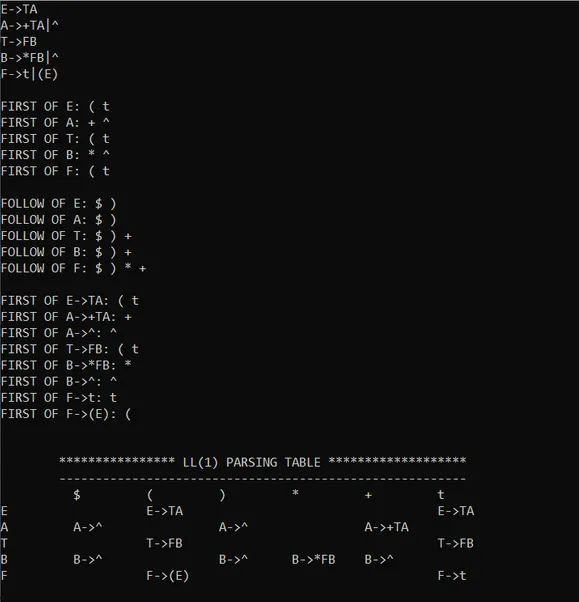
printf("\n");

}

}

}

Output:



**Aim: Develop an operator precedence parser (Construct parse table also).**

Program:

**Aim: Develop a recursive descent parser.**

Program:

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

#define SIZE 128

#define NONE -1

#define EOS '\0'

#define NUM 257

#define KEYWORD 258

#define ID 259

#define DONE 260

#define MAX 999

char lexemes[MAX];

char buffer[SIZE];

int lastchar=-1;

int lastentry=0;

int tokenval=DONE;

int lineno=1;

int lookahead;

struct entry

{

char \*lexptr;

int token;

}symtable[100];

struct entry keywords[]={"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,

"int",KEYWORD,"float",KEYWORD,"double",KEYWORD,

"char",KEYWORD,"struct",KEYWORD,"return",KEYWORD,

0,0};

void errormsg(char \*m)

{

fprintf(stderr,"line %d:%s\n",lineno,m);

exit(1);

}

int lookup(char s[])

{

int k;

for(k=lastentry;k>0;k=k-1)

if(strcmp(symtable[k].lexptr,s)==0)

return k;

return 0;

}

int insert(char s[],int tok)

{int len;

len=strlen(s);

if(lastentry+1>=MAX)

errormsg("symtable is full");

if(lastentry+len+1>=MAX)

errormsg("lexemes array is full");

lastentry=lastentry+1;

symtable[lastentry].token=tok;

symtable[lastentry].lexptr=&lexemes[lastchar+1];

lastchar=lastchar+len+1;

strcpy(symtable[lastentry].lexptr,s);

return lastentry;

}

void initialise()

{

struct entry \*ptr;

for(ptr=keywords;ptr->token;ptr++)

insert(ptr->lexptr,ptr->token);

}

int lexer()

{

int t;

int val,i=0;

while(1)

{

t=getchar();

if(t==' '||t=='\t');

else if(t=='\n')

lineno=lineno+1;

else if(isdigit(t))

{

ungetc(t,stdin);

scanf("%d",&tokenval);

return NUM;

}

else if(isalpha(t))

{

while(isalnum(t))

{ buffer[i]=t;

t=getchar();

i=i+1;

if(i>=SIZE)

errormsg("compile error");

}

buffer[i]=EOS;

if(t!=EOF)

ungetc(t,stdin);

val=lookup(buffer);

if(val==0)

val=insert(buffer,ID);

tokenval=val;

return symtable[val].token;

}

else if(t==EOF)return DONE;

else

{ tokenval=NONE;

return t;

}

}

}

void match(int t)

{

if(lookahead==t)

lookahead=lexer();

else

errormsg("syntax error");

}

void display(int t,int tval)

{

if(t=='+'||t=='-'||t=='\*'||t=='/')

printf("\n arithmetic operator %c",t);

else if(t==NUM)

printf("\n number %d",tval);

else if(t==ID)

printf("\n identifier:%s",symtable[tval].lexptr);

else

printf("\n token:%d tokenval %d",t,tokenval);

}

void F()

{ void E();

switch(lookahead)

{

case '(':

match('(');

E();

match(')');

break;

case NUM:

display(NUM,tokenval);

match(NUM);

break;

case ID:

display(ID,tokenval);

match(ID);

break;

default:

errormsg("syntax error");

}

}

void T()

{ int t;

F();

while(1)

{ switch(lookahead){

case '\*':

t=lookahead;

match(lookahead);

F();

display(t,NONE);

continue;

case '/':

t=lookahead;

match(lookahead);

F();

display(t,NONE);

continue;

default: return;

}

}

}

void E()

{ int t;

T();

while(1)

{ switch(lookahead)

{

case '+':

t=lookahead;

match(lookahead);

T();

display(t,NONE);

continue;

case '-':

t=lookahead;

match(lookahead);

T();

display(t,NONE);

continue;

default:

return;

} } }

void parser()

{

lookahead=lexer();

while(lookahead!=DONE)

{

E();

match(';');

}

}

int main()

{

char ans;

clrscr();printf("\n \t \t Program for recursive decent parsing");

initialise();

printf("enter the expression & place;at the end \n Press CTRL

+ Z to terminate");

parser();

return 0;

}

OUTPUT:

Program for recursive decent parsing

Enter the expression & place ; at the end

Press CTRL + Z to terminate

2+3\*4;

number 2

number 3

number 4

arithmetic operator \*

arithmetic operator +

2+3\*4+5;

number 2

number 3

number 4

arithmetic operator \*

arithmetic operator +

number 5

arithmetic operator +

a-b;

identifier a

identifier b

arithmetic operator –

+1

Line7: syntaxerror