

## LP LAB ASSIGNMENT 8

**amit kumar meena - 207206**

**Q3.**

Write a program in YACC to generate syntax tree, three address code and DAG for a given expression.

**CODE :-**

**LEX:**

```
%option noyywrap
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "3adr.tab.h"
void yyerror(char *);
extern YYSTYPE yylval;
%}
NAME [a-zA-Z]
DIGITS [0-9]+
DOUBLE {DIGITS}(\.{DIGITS})?
%%
[ \t]+ { }
{DOUBLE} {
```

```
strcpy(yylval.cvar, yytext);
return DOUBLE;
}
[-+*/=] {
return *yytext;
}
"(" {
return *yytext;
}
")" {
return *yytext;
}
{NAME} {
strcpy(yylval.cvar, yytext);
return NAME;
}
\n {
return *yytext;
}
exit {
return 0;
}
. {
char msg[25];
sprintf(msg, "<%s> invalid character", yytext);
yyerror(msg);
}
%%
```

# YACC:

```
%{  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
#include <unistd.h>  
int yylex(void);  
int t_count = 1;  
char *str;  
void yyerror(char *s) {  
    fprintf(stderr, "%s \n", s);  
    return;  
}  
char* getTemp(int i) {  
    char *ret = (char*) malloc(15);  
    sprintf(ret, "t%d", i);  
    return ret;  
}  
%}  
%union { char cvar[5]; }  
%token <cvar> DOUBLE  
%token <cvar> NAME  
%token '\n'  
%type <cvar> expr  
%type <cvar> term  
%right '='  
%left '+' '-'
```

```

%left '*'
%left '/'
%left '(' ')'
%%
program:
line program { }
| line { }
;
line:
expr '\n' {
t_count = 1;
printf("\t", "\n", $1);
}
| NAME '=' expr '\n' {
t_count -= 1;
str = getTemp(t_count);
strcpy($3, str);
printf("%s = %s\n", $1, $3);
t_count = 1;
}
;
expr:
expr '+' expr {
str = getTemp(t_count);
strcpy($$, str);
printf("%s = %s + %s\n", $$, $1, $3);
t_count++;
}
| expr '-' expr {
strcpy($$, getTemp(t_count));

```

```

t_count++;
printf("%s = %s - %s \n", $$, $1, $3);
}
| expr '*' expr {
strcpy($$, getTemp(t_count));
t_count++;
printf("%s = %s * %s \n", $$, $1, $3);
}
| expr '/' expr {
strcpy($$, getTemp(t_count));
t_count++;
printf("%s = %s / %s \n", $$, $1, $3);
}
| term {
strcpy($$, $1);
}
| '(' expr ')' {
strcpy($$, getTemp(t_count));
t_count++;
printf("%s = (%s) \n", $$, $2);
}
;
term:
NAME {
strcpy($$, $1);
}
| DOUBLE {
strcpy($$, $1);
}
;

```

```
%%  
int main(void) {  
    yyparse();  
    return 0;  
}
```

**output:-**

```
a+(b*c)+(e*f)  
t1 = b * c  
t2 = (t1)  
t3 = a + t2  
t4 = e * f  
t5 = (t4)  
t6 = t3 + t5
```

## Q4.

Write a program to generate a symbol table using words in a given English text. Also include rules to recognise words like “can’t” as a single word. Use a normal table for storing words, their frequency and unique line numbers in which the word appears.

## CODE :-

```
%option noyywrap yylineno nodefault case-insensitive
%{
    struct ref
    {
        char* filename;
        int line;
        struct ref* next;
    };
    struct Symtab
    {
        char* word;
        struct ref* ptr;
        int num;
    };
    #define NHASH 211
    struct Symtab hashtable[NHASH];
    int hash(char* word);
    int insert(char* word);
    char* curr_file;
    void add_end(struct ref** ptr,char* filename,int line);
```

```

        void init();
        void show();
        int num_elements;
    %}

%%

a |
an |
are |
is |
was |
were |
in |
the |
up |
at |
on |
am |
down |
there |
this |
that |
those |
these |
between |
near |
to ;
[a-z]+(\\'(s|t))? { insert(yytext);}
<<EOF>> { yyterminate();}
\\n ;
. ;

%%

```



```

int hash(char* word)
{
    int n = strlen(word), h = 0;
    for(int i=0; i<n; i++)
    {
        h = (h*10+((int)(word[i])*(i+1))%NHASH)%NHASH;
    }
    return h;
}

```

```

void add_end(struct ref** ptr, char* filename, int line)
{
    if(*ptr==NULL)
    {
        *ptr = (struct ref*)malloc(sizeof(struct ref));
        (*ptr)->filename = NULL;
        (*ptr)->filename = (char*)malloc(sizeof(char));
        strcpy((*ptr)->filename, filename);
        (*ptr)->line = line;
        (*ptr)->next = NULL;
    }
    else
    {
        struct ref* q;
        q = *ptr;
        while(q->next!=NULL)
        {
            q = q->next;
        }
        q->next = (struct ref*)malloc(sizeof(struct ref));
        q->next->filename = NULL;
        q->next = (struct ref*)malloc(sizeof(struct ref));
        q->next->filename = (char*)malloc(sizeof(char));
        strcpy(q->next->filename, filename);
        q->next->line = line;
    }
}

```

```

        q->next->next = NULL;
    }
}

int insert(char* word)
{
    if(num_elements==NHASH)
    {
        printf("No Space available\n");
        yyterminate();
    }
    int h = hash(word),f = 0;
    while(hashtab[h].ptr!=NULL)
    {
        if(hashtab[h].word!=NULL&&strcmp(hashtab[h].word,word)==0)
        {
            f = 1;
            break;
        }
        h = (h+1)%NHASH;
    }
    hashtab[h].num++;
    add_end(&hashtab[h].ptr,curr_file,yylineno);
    if(!f)
    {
        hashtab[h].word = NULL;
        hashtab[h].word = (char*)malloc(sizeof(char));
        strcpy(hashtab[h].word,word);
        num_elements++;
    }
    return 1;
}

void init()
{
    for(int i=0;i<NHASH;i++)

```

```

    {
        hashtab[i].word = NULL;
        hashtab[i].ptr = NULL;
        hashtab[i].num = 0;
    }
    num_elements = 0;
}

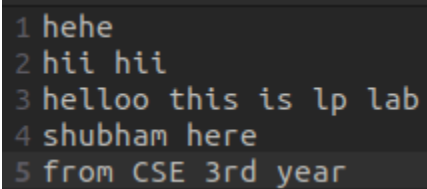
void show()
{
    for(int i=0;i<NHASH;i++)
    {
        if(hashtab[i].num!=0)
        {
            printf("%s : %d\n",hashtab[i].word,hashtab[i].num);
            struct ref* q = hashtab[i].ptr;
            while(q!=NULL)
            {
                printf("      %s : %d\n",q->filename,q->line);
                q = q->next;
            }
        }
    }
}

int main(int argc,char* argv[])
{
    init();
    curr_file = NULL;
    curr_file = (char*)malloc(sizeof(char));
    if(argc==1)
    {
        strcpy(curr_file,"STDIN");
        yylex();
    }
    else
    {

```

```
for(int i=1;i<argc;i++)
{
    yyin = fopen(argv[i],"r");
    if(yyin==NULL)
    {
        perror("File not opened");
    }
    else
    {
        strcpy(curr_file,argv[i]);
        yylex();
    }
}
show();
return 0;
}
```

## Text File :-



```
1 hehe
2 hii hii
3 helloo this is lp lab
4 shubham here
5 from CSE 3rd year
```

# OUTPUT :-

```
year : 1
      file.txt : 5
lp : 1
      file.txt : 3
helloo : 1
        file.txt : 3
from : 1
      file.txt : 5
hehe : 1
      file.txt : 1
rd : 1
      file.txt : 5
shubham : 1
          file.txt : 4
CSE : 1
      file.txt : 5
hii : 2
      file.txt : 2
      file.txt : 2
here : 1
      file.txt : 4
lab : 1
      file.txt : 3
```

## Q5.

Write a program to find LR(0) items for the following expression grammar and construct SLR table assuming that the operators '+' and '\*' are right associative and + has higher precedence than \*

$$E \rightarrow E + E \mid E * E \mid (E) \mid id$$

## CODE :-

```
#include<bits/stdc++.h>
#include<unistd.h>
#include<fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
using namespace std;
struct slrEntry
{
    map<char,vector<string> > action;
    map<char,int> go2;
};
vector<slrEntry> table;
vector<vector<string> > grammar(26);
map<vector<vector<string> >,pair<int,bool> > itsets; //lr 0 item sets
int cntitsets = 0;
vector<set<char> > first(26),follow(26);

void printGr()
{
```

```

for(int j = 0; j < 26; j++)
{
    for(int i=0;i<grammar[j].size();i++)
        cout<<(char)(j+'A')<<" -> "<<grammar[j][i]<<"\n";
}

}

void printItsets(){
    for(map<vector<vector<string> >,pair<int,bool> >::iterator
it=itsets.begin();it!=itsets.end();it++){
        cout<<"Itemset "<<(it->second).first<<":\n";
        for(int i=0;i<26;i++){
            for(int j = 0; j < (it->first)[i].size(); j++){
                cout<<(char)(i+'A')<<" -> "<<(it->first)[i][j]<<"\n";
            }
        }
        cout<<"\n";
    }
}

int findDot(string alp,string &befDot,string &aftDot)
{
    int pos = -1;
    for(int i=0;i<alp.size();i++)
    {
        if(alp[i] == '.')
        {
            pos = i;
            if(i+1<alp.size())
                befDot+=alp[i+1];
            i++;
            continue;
        }
        if(pos == -1)
        {

```

```

        befDot+=alp[i];
    }
    else
    {
        aftDot+=alp[i];
    }
}
return pos;
}

```

```

void closure(vector<vector<string> > &curr)
{
    bool f = 1;
    bool vis[26] = {0};
    while(f)
    {
        f = 0;
        for(int i=0;i<26;i++)
        {
            if(curr[i].size()>0)
            {
                for(int j=0;j<curr[i].size();j++)
                {
                    string prod = curr[i][j],ad,bd;
                    int pos = findDot(prod,bd,ad);
                    if(pos+1<prod.size())
                    {
                        char x = prod[pos+1];
                        if(isupper(x) && !vis[x-'A'])
                        {
                            int nidx = x-'A';

                            vis[nidx] = 1;
                            for(int k=0;k<grammar[nidx].size();k++)

```



```
{
curr[nidx].push_back("."+grammar[nidx][k]),
    }
    f = 1;
}
}
}
}
}
```

```

void genNewItset(vector<vector<string> > &curr,char
x,vector<vector<string> > &nitset)
{
    for(int i=0;i<26;i++){
        if(curr[i].size()>0){
            for(int j=0;j<curr[i].size();j++){
                string prod = curr[i][j],bd,ad;
                int pos = findDot(prod,bd,ad);
                if(pos == prod.size()-1)
                    continue;
                if(prod[pos+1] == x){
                    string nextrhs = bd+"."+ad;

                    nitset[i].push_back(nextrhs);
                }
            }
        }
    }
}

void addElems(set<char> &a,set<char> &b)
{

```

```

    for(set<char>::iterator it = b.begin();it!=b.end();it++)
    {
        a.insert((*it));
    }
}
void computeFirst(int v)
{
    for(int i=0;i<grammar[v].size();i++)
    {
        if(isupper(grammar[v][i][0]))
        {
            if((grammar[v][i][0]-'A' == v))
            {
                continue;
            }
            if(first[grammar[v][i][0]-'A'].size()==0)
                computeFirst(grammar[v][i][0]-'A');
            addElems(first[v],first[grammar[v][i][0]-'A']);
        }
        else
        {
            first[v].insert(grammar[v][i][0]);
        }
    }
}
void computeFollow(int v)
{
    char x = v+'A';
    for(int i=0;i<26;i++)
    {
        if(grammar[i].size()>0)

```

```

{
    for(int j=0;j<grammar[i].size();j++)
    {
        string prod = grammar[i][j];
        for(int k=0;k<prod.size();k++)
        {
            if(prod[k] == x)
            {
                if(k == prod.size()-1)
                {
                    if(follow[i].size()==0)
                        computeFollow(i);
                    addElems(follow[v],follow[i]);
                }
            }
            else
            {
                if(isupper(prod[k+1]))
                {
                    addElems(follow[v],first[prod[k+1]-'A']);
                }
            }
            else
            {
                follow[v].insert(prod[k+1]);
            }
        }
    }
}

void read(vector<vector<string> > &curr,bool &end,int tableindex)
{
    int f;

```

```

set<char> vis;
for(int i=0;i<26;i++)
{
    if(curr[i].size(>0)
    {
        for(int j=0;j<curr[i].size();j++)
        {
            string prod = curr[i][j],bd,ad;
            f = 1;
            int pos = findDot(prod,bd,ad);
            if(pos == prod.size()-1)
            {
                for(set<char>::iterator
it=follow[i].begin();it!=follow[i].end();it++)
                {
                    if((*it) == '$' && i==25)

                    {
                        table[tableindex].action[(*it)].push_back("acc");
                    }
                    else
                    {
                        string ent = "R ";
                        ent+=((char)(i+'A'));
                        ent+="->";
                        ent+=bd;
                        cout<<"Reduce entry of itemset
"<<tableindex<<": "<<ent<<"\n";

                        table[tableindex].action[(*it)].push_back(ent);
                    }
                }
                continue;
            }
            char x = prod[pos+1];

```

```

        if(vis.find(x) == vis.end())
        {
            vector<vector<string> > nit(26);
            genNewItset(curr,x,nit);
            vis.insert(x);
            closure(nit);
            map<vector<vector<string> >,pair<int,bool>
>::iterator it = itsets.find(nit);
            int transitset;
            if(itsets.find(nit) == itsets.end())
            {

                transitset = cntitsets;

itsets.insert(make_pair(nit,make_pair(cntitsets++,0)));
                slrEntry e;
                table.push_back(e);
                end = 1;

            }
        else
            transitset = (it->second).first;
            if(isupper(x))
            {

table[tableindex].go2.insert(make_pair(x,transitset));
            }
        else
        {

            string tmp="S";
            tmp+=to_string(transitset);
            table[tableindex].action[x].push_back(tmp);

```

```

                                cout<<"Shift entry of itemset
" << tableindex << ": " << tmp << "\n";
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

void printFirst(){
    for(int i=0;i<26;i++){
        if(first[i].size()>0){
            cout<<"First(" << (char)(i+'A') << ") = ";
            for(set<char>::iterator it = first[i].begin();it!=first[i].end();it++)
                cout<<(*it)<<" ";
            cout<<"\n";
        }
    }
}

void printFollow(){
    for(int i=0;i<26;i++){
        if(follow[i].size()>0){
            cout<<"Follow(" << (char)(i+'A') << ") = ";
            for(set<char>::iterator it =
follow[i].begin();it!=follow[i].end();it++)
                cout<<(*it)<<" ";
            cout<<"\n";
        }
    }
}

void printTable()
{
    for(int i=0;i<table.size();i++)
    {
        cout<<"State " << i << "\n";
        cout<<"action:\n";
    }
}

```

```

        for(map<char,vector<string> >::iterator
it=table[i].action.begin();it!=table[i].action.end();it++){
            cout<<"("<<it->first<<" , ";
            for(int j=0;j<(it->second).size();j++)
                cout<<(it->second)[j]<<" ";
            cout<<"\n";
        }
        cout<<"goto:\n";
        for(map<char,int>::iterator
it=table[i].go2.begin();it!=table[i].go2.end();it++){
            cout<<"("<<it->first<<" , "<<(it->second)<<"\n";
        }
        cout<<"\n";
    }
}

```

```

int findOp(string s)

```

```

{
    for(int i=0;i<s.size();i++)
    {
        if(s[i] == '+' || s[i] == '*')
            return i;
    }
    return -1;
}

```

```

//assoc = 1, implies right ; prec = 1, implies + > *

```

```

void resolveAmbiguity(bool assoc,bool prec)

```

```

{
    for(int i=0;i<table.size();i++)
    {
        for(map<char,vector<string> >::iterator
it=table[i].action.begin();it!=table[i].action.end();it++){
            {
                if((it->second).size()==2)
            {

```

```

        int r,s;
        if((it->second)[0][0] == 'R')
        {
            r = 0;
            s = 1;
        }
        else
        {
            r = 1;
            s = 0;
        }
        int opPos = findOp((it->second)[r]);
        if((it->first) == (it->second)[r][opPos])
        {
            //in this case, since operators are same, we have to resolve the
associativity
            if(assoc)
            {
                (it->second).erase((it->second).begin()+r);
            }
            else
            {
                (it->second).erase((it->second).begin()+s);
            }
        }
        else
        {
            if(prec)
            {
                if((it->first == '+') &&
(it->second)[r][opPos]=='*')
                {
                    (it->second).erase((it->second).begin()+r);
                }
            }
        }
    }
}

```



```

        if((it->first == '*') &&
(it->second)[r][opPos]=='+')
            {
                (it->second).erase((it->second).begin()+s);
            }
        else
        {
            if((it->first == '+') &&
(it->second)[r][opPos]=='*')
                {
                    (it->second).erase((it->second).begin()+s);
                }
            if((it->first == '*') &&
(it->second)[r][opPos]=='+')
                {
                    (it->second).erase((it->second).begin()+r);
                }
            }
        }
    }
}

int main()
{
    int infd = open("input.txt",O_RDONLY);
    dup2(infd,0);

    int n;
    cin>>n;
    char start='.';

```

```

for(int i=0;i<n;i++)
{
    string s;
    cin>>s;
    if(start == '.')
    {
        start = s[0];
    }
    string prod;
    bool f = 0;
    for(int i=0;i<s.length();i++)
    {
        if(s[i] == '-' && s[i+1] == '>')
        {
            f = 1;
            i++;
            continue;
        }
        if(f)
            prod+=s[i];
    }
    int idx = s[0]-'A';
    grammar[idx].push_back(prod);
}
cout<<"Augmented Grammar\n";
fflush(stdout);
string rhs;
rhs+=start;
grammar[25].push_back(rhs);
printGr();
vector<vector<string> > is0(26);

is0[25].push_back("."+rhs);
closure(is0);
itsets.insert(make_pair(is0,make_pair(cntitsets++,0)));

```

```

slrEntry e0;
table.push_back(e0);
//computing first of all variables
for(int i=0;i<26;i++)
{
    if(grammar[i].size()>0 && first[i].size()==0)
    {
        computeFirst(i);
    }
}
printFirst();

follow[25].insert('$');
//computing follow of all variables in the grammar
for(int i=0;i<26;i++)
{
    if(grammar[i].size()>0 && follow[i].size()==0)
    {
        computeFollow(i);
    }
}
printFollow();
//printItsets();
bool f = 1;
while(f)
{
    f = 0;
    for(map<vector<vector<string> >,pair<int,bool> >::iterator
it=itsets.begin();it!=itsets.end();it++)
    {

        if((it->second).second == 0)
        {
            vector<vector<string> > is = it->first;
            int idx = (it->second).first;

```

```

        read(is,f,idx);
        (it->second).second = 1;
    }
}

}
cout<<"Itemsets are\n";
printItsets();
resolveAmbiguity(1,1);
cout<<"-----> SLR Parsing Table <-----\n";

printTable();
return 0;
}

```

# OUTPUT :-

```
Augmented Grammar
E -> E+E
E -> E*E
E -> (E)
E -> i
Z -> E
First(E) = ( i
First(Z) = ( i
Follow(E) = $ ) * +
Follow(Z) = $
Shift entry of itemset 0: S2
Shift entry of itemset 0: S3
Shift entry of itemset 1: S4
Shift entry of itemset 1: S5
Reduce entry of itemset 3: R E->i
Reduce entry of itemset 3: R E->i
Reduce entry of itemset 3: R E->i
Reduce entry of itemset 3: R E->i
Shift entry of itemset 2: S2
Shift entry of itemset 2: S3
Shift entry of itemset 6: S7
Shift entry of itemset 6: S4
Shift entry of itemset 6: S5
Shift entry of itemset 5: S2
Shift entry of itemset 5: S3
Reduce entry of itemset 8: R E->E*E
Reduce entry of itemset 8: R E->E*E
Reduce entry of itemset 8: R E->E*E
Reduce entry of itemset 8: R E->E*E
Shift entry of itemset 8: S4
Shift entry of itemset 8: S5
Shift entry of itemset 4: S2
Shift entry of itemset 4: S3
Reduce entry of itemset 9: R E->E+E
Reduce entry of itemset 9: R E->E+E
Reduce entry of itemset 9: R E->E+E
Reduce entry of itemset 9: R E->E+E
Shift entry of itemset 9: S4
Shift entry of itemset 9: S5
Reduce entry of itemset 7: R E->(E)
Reduce entry of itemset 7: R E->(E)
Reduce entry of itemset 7: R E->(E)
Reduce entry of itemset 7: R E->(E)
```

Itemsets are

Itemset 2:

$$E \rightarrow (.E)$$

E -> .E+E

$$E \rightarrow .E * E$$
$$E \rightarrow \cdot (E)$$
$$E \rightarrow \cdot i$$

1. *Chlorophyll a* (Chl *a*)

$$E \rightarrow (E).$$

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

$$E \rightarrow (E.)$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$

1. *Journal of the American Medical Association*, 1997; 277: 1033-1037.

$$E \rightarrow \cdot E + E$$
$$E \rightarrow \cdot E * E$$
$$E \rightarrow \cdot (E)$$
$$E \rightarrow \cdot i$$
$$7 \rightarrow E$$

2.  $\rightarrow$  .E

$$E \rightarrow E^* . E$$

E  $\rightarrow$  .E+E

$$E \rightarrow \cdot E^*E$$
$$E \rightarrow \cdot (F)$$
$$E \rightarrow i$$

E → .L

$$E \rightarrow E * E.$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$

**L**

$$E \rightarrow E + .E$$

E -> .E+E

$$E \rightarrow \cdot E * E$$
$$E \rightarrow \cdot (E)$$
$$E \rightarrow \cdot i$$

$E \rightarrow \cdot U$

$$E \rightarrow E + E.$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$

1. *Journal of Management Studies*, 1990, 27, 1, 1-14.

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$Z \rightarrow E.$$

2.  $\angle A = \angle C$ .

$$E \rightarrow i.$$

TABLE 1

```
-----> SLR Parsing Table <-----  
State 0  
action:  
((, S2 )  
(i, S3 )  
goto:  
(E,1)  
  
State 1  
action:  
($, acc )  
(*, S5 )  
(+, S4 )  
goto:  
  
State 2  
action:  
((, S2 )  
(i, S3 )  
goto:  
(E,6)  
  
State 3  
action:  
($, R E->i )  
( ), R E->i )  
(*, R E->i )  
(+, R E->i )  
goto:  
  
State 4  
action:  
((, S2 )  
(i, S3 )  
goto:  
(E,9)  
  
State 5  
action:  
((, S2 )  
(i, S3 )  
goto:  
(E,8)
```

State 6

action:

( ), S7 )

( \*, S5 )

( +, S4 )

goto:

State 7

action:

( \$, R E  $\rightarrow$  (E) )

( ), R E  $\rightarrow$  (E) )

( \*, R E  $\rightarrow$  (E) )

( +, R E  $\rightarrow$  (E) )

goto:

State 8

action:

( \$, R E  $\rightarrow$  E \* E )

( ), R E  $\rightarrow$  E \* E )

( \*, S5 )

( +, S4 )

goto:

State 9

action:

( \$, R E  $\rightarrow$  E + E )

( ), R E  $\rightarrow$  E + E )

( \*, R E  $\rightarrow$  E + E )

( +, S4 )

goto: