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);
}
%%
```

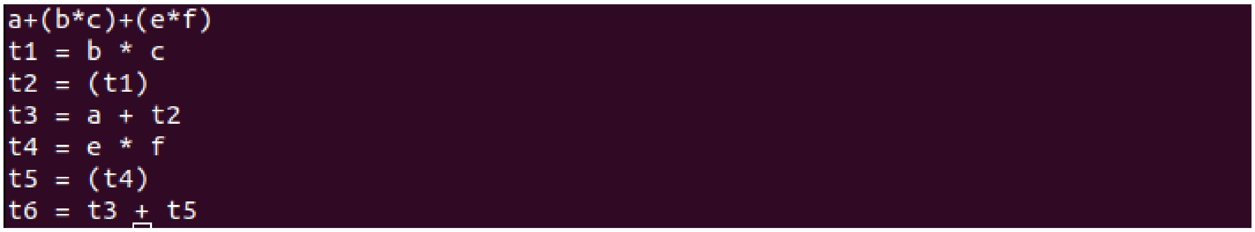
YACC:

```
%{
#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.

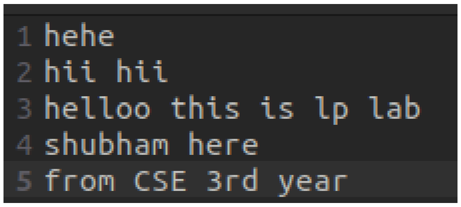
CODE :-

```
%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 hashtab[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++)
```

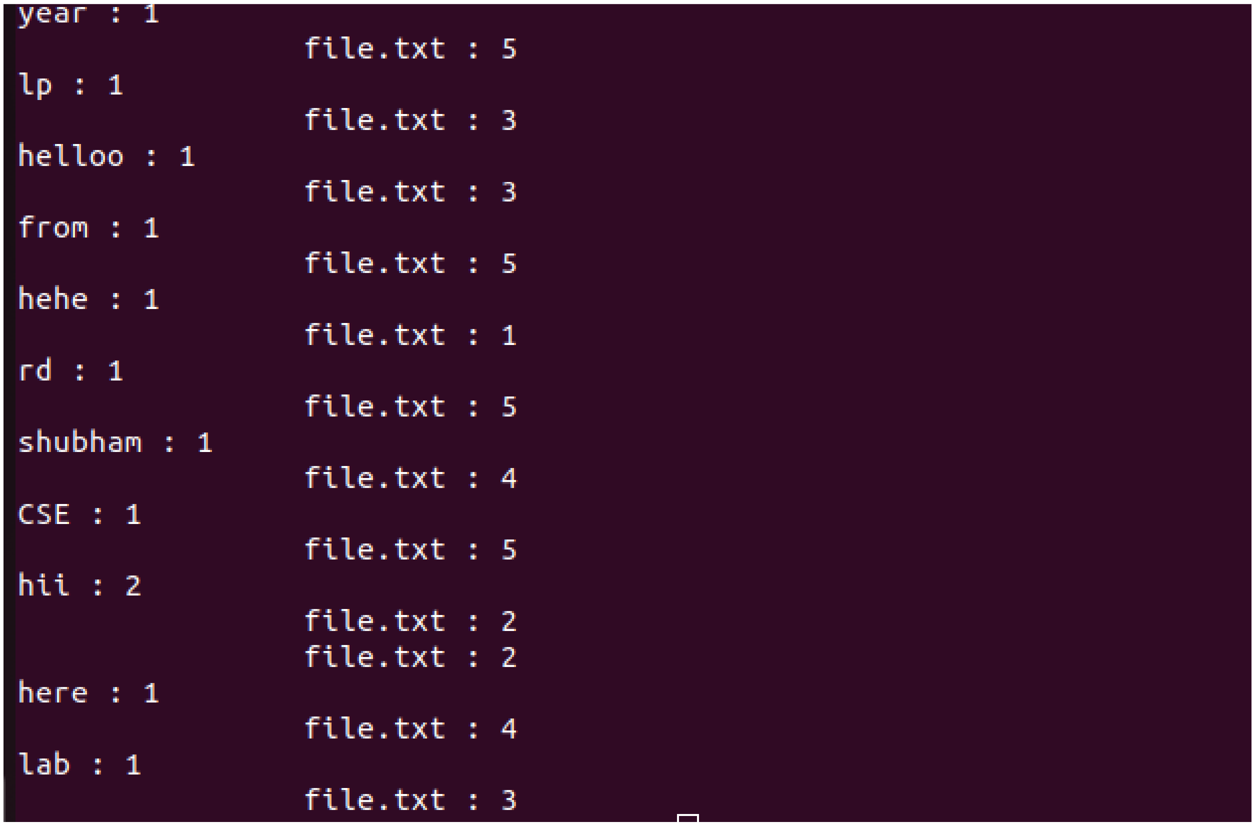
```
    {
        hashtab[i].word = NULL;
        hashtab[i].ptr = NULL;
        hashtab[i].num = 0;
    }
    num_elements = 0;
}
void show()
{
    for(int i=0;i<NHASH;i++)
    {
        if(hashtab[i].num!=0)
        {
            printf("%s : %d\n",hashtab[i].word,hashtab[i].num);
            struct 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 :-



```
hehe
hii hii
helloo this is lp lab
shubham here
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 → E + E | E * E | (E) | 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";
```

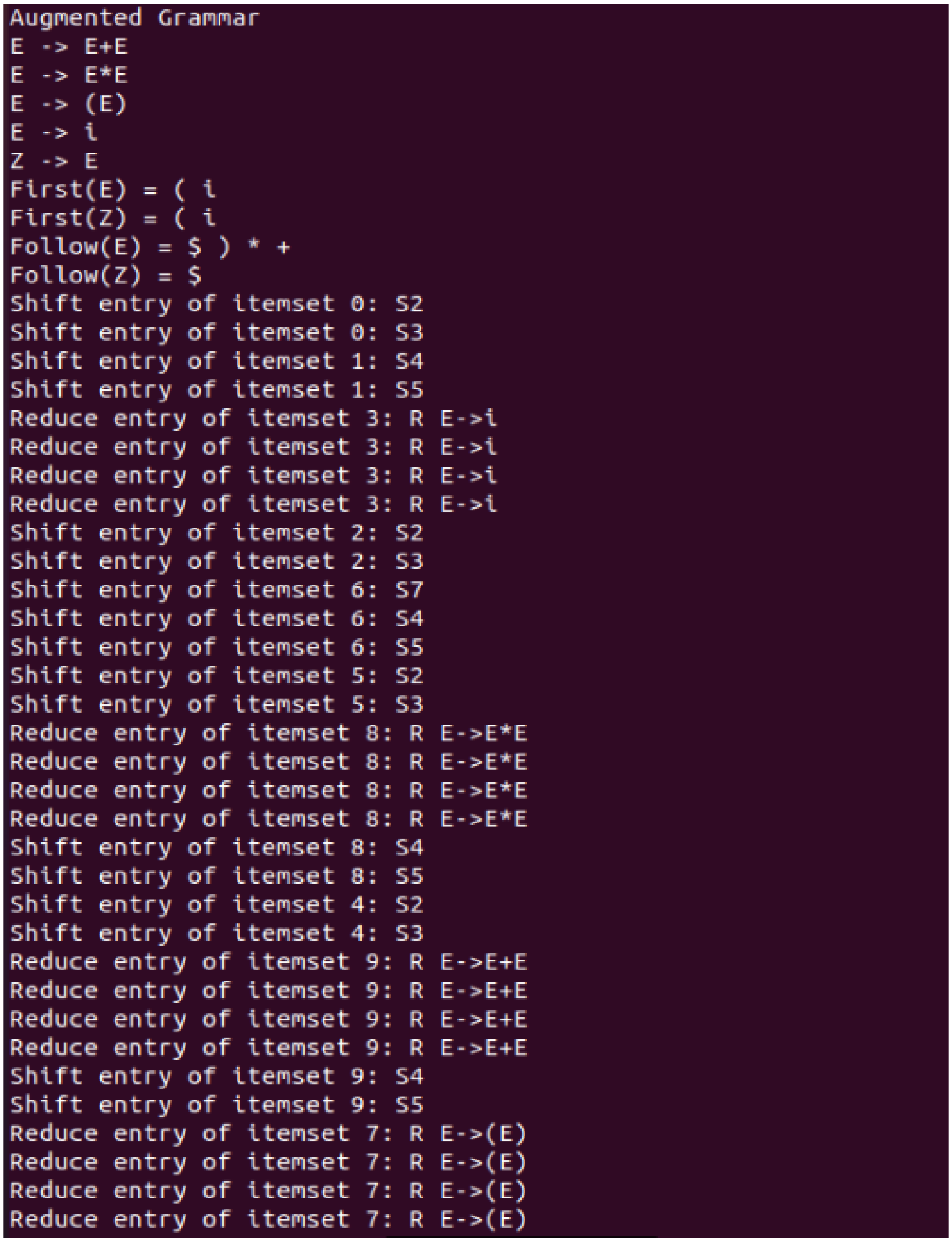
```
          for(map<char,vector<string> >::iterator
it=table[i].action.begin();it!=table[i].action.end();it++){
                    cout<<"("<<it->first<<", ";
                    for(int j=0;j<(it->second).size();j++)
                              cout<<(it->second)[j]<<" ";
                    cout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

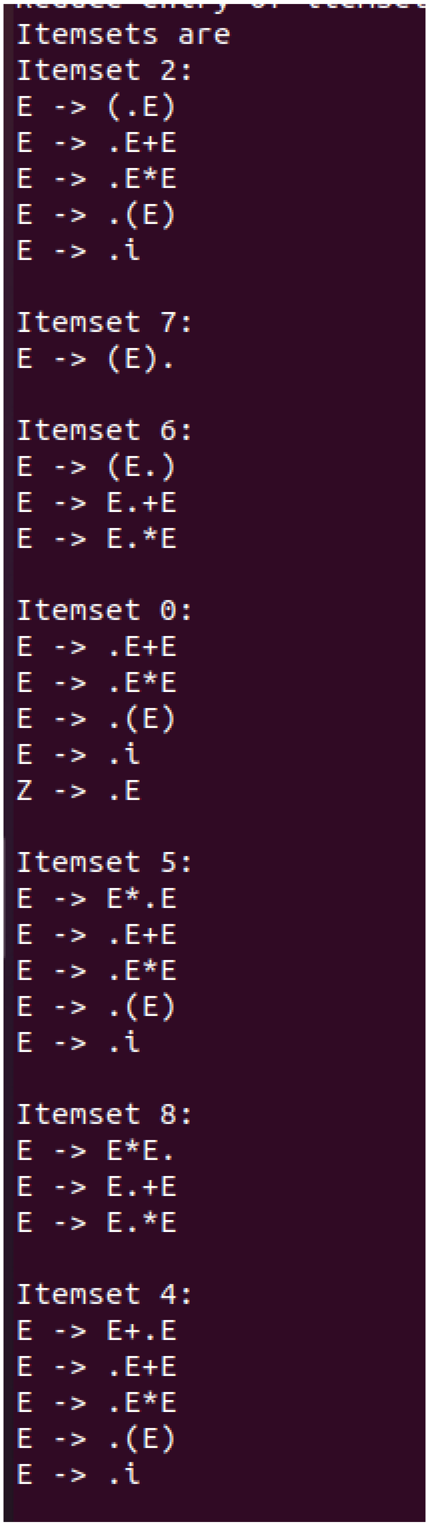
```
for(int i=0;i<n;i++)
    {
     string s;
     cin>>s;
     if(start == '.')
    {
            start = s[0];
     }
     string prod;
     bool f = 0;
     for(int i=0;i<s.length();i++)
    {
            if(s[i] == '-' && s[i+1] == '>')
    {
                    f = 1;
                    i++;
                    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;
}
```

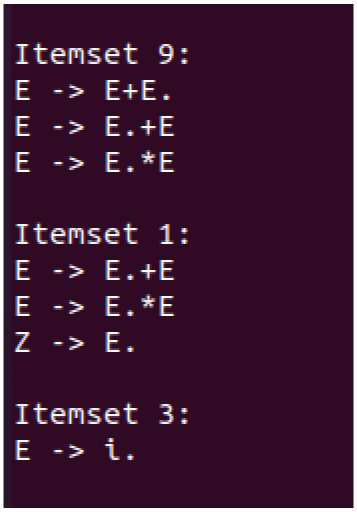
OUTPUT :-



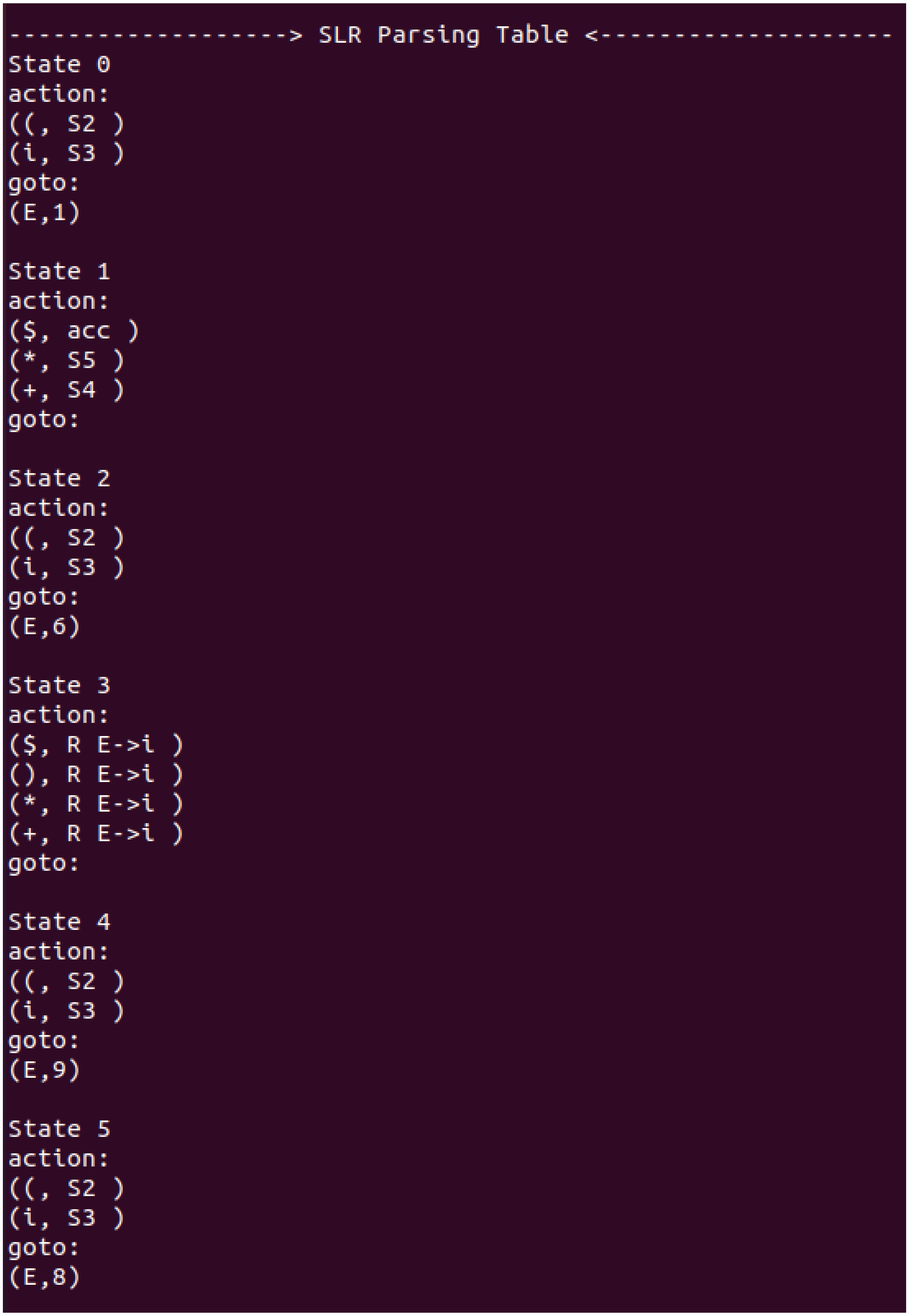
```
Augmented Grammar
E -> E+E
E -> E*E
E -> (E)
E -> i
Z -> E
First(E) = ( i
First(Z) = ( i
Follow(E) = $ ) * +
Follow(Z) = $
Shift entry of itemset 0: S2
Shift entry of itemset 0: S3
Shift entry of itemset 1: S4
Shift entry of itemset 1: S5
Reduce entry of itemset 3: R E->i
Reduce entry of itemset 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)
```



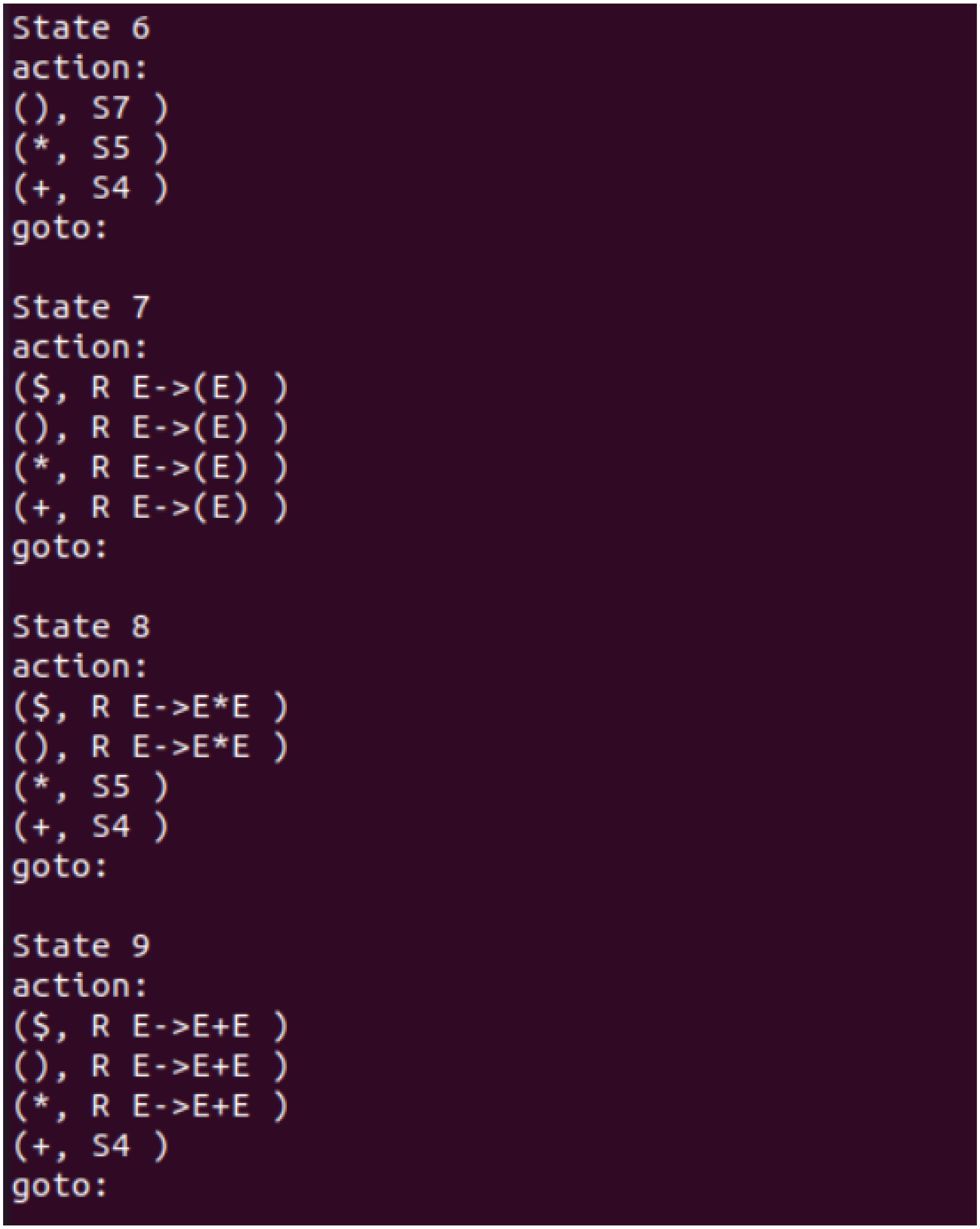
```
Itemsets are
Itemset 2:
E -> (.E)
E -> .E+E
E -> .E*E
E -> .(E)
E -> .i

Itemset 7:
E -> (E).

Itemset 6:
E -> (E.)
E -> E.+E
E -> E.*E

Itemset 0:
E -> .E+E
E -> .E*E
E -> .(E)
E -> .i
Z -> .E

Itemset 5:
E -> E*.E
E -> .E+E
E -> .E*E
E -> .(E)
E -> .i

Itemset 8:
E -> E*E.
E -> E.+E
E -> E.*E

Itemset 4:
E -> E+.E
E -> .E+E
E -> .E*E
E -> .(E)
E -> .i
```



```
Itemset 9:
E -> E+E.
E -> E.+E
E -> E.*E

Itemset 1:
E -> E.+E
E -> E.*E
Z -> E.

Itemset 3:
E -> i.
```



```
-------------------> 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->(E) )
(), R E->(E) )
(*, R E->(E) )
(+, R E->(E) )
goto:

State 8
action:
($, R E->E*E )
(), R E->E*E )
(*, S5 )
(+, S4 )
goto:

State 9
action:
($, R E->E+E )
(), R E->E+E )
(*, R E->E+E )
(+, S4 )
goto:
```