LL(1) Table

```
#include<iostream>
#include<map>
#include<vector>
#include<string>
#include<algorithm>
#include<set>
#include<iomanip>
using namespace std;

void create_prod(string);
string getFirst(char);
string getFollow(char);
void getLL(char);
void printLLTable();
map<char,vector<string> > prod;
map<char,string > first,follow;
int n;

char *order;
map<char,string> terminals;
map<char,string> :: iterator itr;

int main()
{
		cout<<"Enter the number of productions: ";
		cin>>n;
		order=new char[n];
		cout<<"Enter the productions:\n";
		for(int i=0;i<n;i++)
		{
		   string s;
		   cin>>s;
		   order[i]=s[0];
		   create_prod(s);
		}
		for(int i=0;i<n;i++)
		{
		   first[order[i]]=getFirst(order[i]);
		   cout<<"FIRST("<<order[i]<<") = "<<first[order[i]]<<endl;
		}
		for(int i=0;i<n;i++)
		{
		   follow[order[i]]=getFollow(order[i]);
		   cout<<"FOLLOW("<<order[i]<<") = "<<follow[order[i]]<<endl;
		}
		printLLTable();
		return 0;
}

void create_prod(string s)
{
   int i=3;
   string in="";
   vector<string> v;
   while(i<s.length())
```

```
        if(s[i]=='|')
        {
            v.push_back(in); in=""; i++;
        }
        else    in+=s[i++];
    v.push_back(in);
    prod[s[0]]=v;
}

void printLLTable()
{
    cout<<"\n\t\tLL(1) PARSING TABLE\n";
    terminals['$']="";
    cout<<setw(10)<<"|";
    for(itr=terminals.begin();itr!=terminals.end();itr++)      cout<<left<<setw(10)<<itr->first;
    cout.fill('-');
    cout<<setw((terminals.size()+1)*10)<<"\n";
    cout.fill(' ');
    cout<<endl;
    for(int i=0;i<n;i++)      getLL(order[i]);
    cout<<endl;
}

void getLL(char c)
{
    cout<<left<<setw(9)<<c<<"|";
    vector<string> v=prod[c];
    string fir=first[c],fol=follow[c],ans="";

    for(int i=0;i<fir.length();i++)
    {
                char ch=fir[i];
                if(ch=='e')
                            for(int i=0;i<fol.length();i++)
                              {
                                        ans=ans+c+"->e";
                                        terminals[fol[i]]=ans;
                                        ans="";
                              }
                else{
                            for(int j=0;j<v.size();j++)
                            {
                              string temp=first[v[j][0]];
                              if(temp.find(ch)!=std::string::npos)
                              {
                                        ans=ans+c+"->"+v[j];
                                        terminals[ch]=ans;
                                        ans="";
                              }
                            }
                }
    }
    for(itr=terminals.begin();itr!=terminals.end();itr++) cout<<left<<setw(10)<<terminals[itr->first];
    cout<<endl;
    for(itr=terminals.begin();itr!=terminals.end();itr++) terminals[itr->first]="";
}
```

```
string getFollow(char c)
{
   string ans="";
   if(c==order[0]) ans+="$";
   for(int i=0;i<n;i++)
   {
      char head=order[i];
      vector<string> v=prod[head];
      for(int j=0;j<v.size();j++)
      {
         string temp=v[j];
         int found=temp.find(c);
         if(found!= std::string::npos)
         {
            if(found+1!=temp.length())
            {
               string newtemp=getFirst(temp[found+1]);
               if(newtemp[0]=='e')
               {
                  newtemp=newtemp.substr(1);
                  if(head!=c) newtemp+=getFollow(head);
               }
               ans+=newtemp;
            }
            else    if(head!=c) ans+=getFollow(head);
         }
      }
   }
   //remove duplicates from the answer
   sort(ans.begin(),ans.end());
   string temp=ans;
   ans.resize( std::distance(ans.begin(),std::unique_copy(temp.begin(),temp.end(),ans.begin())));
   return ans;
}

string getFirst(char c)
{
   string ans="";
   if(prod.find(c)==prod.end())
   {
      if(c!='e') terminals[c]="";
      ans+=c;
      first[c]=ans;
      return ans; //terminal
   }
   vector<string> v=prod[c];
   for(int i=0;i<v.size();i++)
   {
      string temp=getFirst(v[i][0]);
      if(temp=="e") ans="e"+ans;
      else ans+=temp;
   }
   return ans;
}
```

3-address Code Generation:

```
#include<iostream>
#include<string>
using namespace std;

string input,op,arg1,arg2;
int l,j=0;
void rep(int);
void checkUnaryMinus(int);
int getPrecedence(char);

void printCode();
string getCode(char);

int main()
{
        int i,p;
        cout<<"Enter the input Expression\n";
        cin>>input;
        l=input.length();
        for(i=0;i<l;i++)    if(input[i]=='-') checkUnaryMinus(i);

        for(int n=6;n>=4;n--)
       {
                for(i=0;i<l;i++)
                {
                        p=getPrecedence(input[i]);
                        if(p==n)
                        {
                                rep(i);     i=0;
                        }
                }
        }

        cout<<"The triplet 3-address code notation is \n";
        cout<<"No:\top\targ1\targ2\n";
        for(i=0;i<j;i++)
             cout<<i<<"\t"<<op[i]<<"\t"<<arg1[i]<<"\t"<<arg2[i]<<endl;

        cout<<"\nGenerated code\n";
        printCode();

}
int getPrecedence(char c)
{
        switch(c){
                case '*':
                case '/':     return 6;
                case '+':
                case '-':     return 5;
                case '=':     return 4;
                default :     return -1;
        }
}
```

```
void rep(int i)
{
        op[j]=input[i];
        arg1[j]=input[i-1];
        arg2[j]=input[i+1];

        input.replace(i-1, 3, to_string(j));
        l=l-2;
        j++;
}

void checkUnaryMinus(int i)
{
        if(!isalpha(input[i-1]))
        {
                op[j]='m';
                arg1[j]=input[i+1];
                arg2[j]=' ';
                input.replace(i, 2, to_string(j));
                l--;
                j++;
        }
}

string getCode(char c)
{
        switch(c){
                case '*':   return "MUL";
                case '/':   return "DIV";
                case '+':   return "ADD";
                case '-':   return "SUB";
        }
}

void printCode()
{
        for(int i=0;i<j;i++)
        {
                if(op[i]!='=')
                {
                        cout<<"MOV "<<arg1[i]<<",R0\n";

                        if(op[i]=='m')      cout<<"NEG "<<arg1[i]<<endl;
                        else                cout<<getCode(op[i])<<" "<<arg2[i]<<" , R0\n";
                }
                else                        cout<<"MOV "<<arg2[i]<<" , "<<arg1[i]<<endl;
        }
}
```

Yacc Simple Calculator
dummy.y

```
%{
            #include <stdio.h>
            int yylex(void);
            void yyerror(char *);
%}
%token INTEGER

%%
program:
               program expr '\n'       { printf("%d\n", $2); }
               |
               ;
expr:
               INTEGER                 { $$ = $1; }
               | expr '+' expr          { $$ = $1 + $3; }
               | expr '-' expr          { $$ = $1 - $3; }
               | expr '*' expr          { $$ = $1 * $3; }
               | expr '/' expr          { $$ = $1 / $3; }
               ;
%%

void yyerror(char *s) {
              fprintf(stderr, "%s\n", s);
}

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

dummy.l

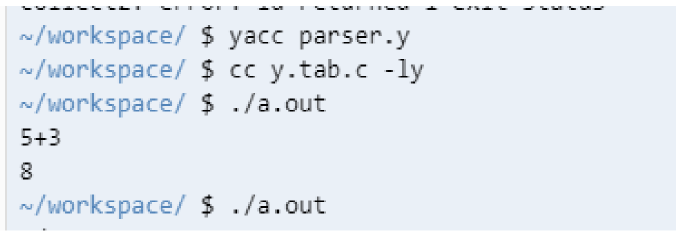
```
%{
            #include "y.tab.h"
            #include <stdlib.h>
            void yyerror(char *);
%}

%%

[0-9]+ {   yylval = atoi(yytext);
           return INTEGER;
        }

[-+*/\n]      return *yytext;
[ \t]     ;           /* skip whitespace */
.          yyerror("invalid character");

%%

int yywrap(void) {
   return 1;
}
```

Parser Generator using YACC:

Parser.y

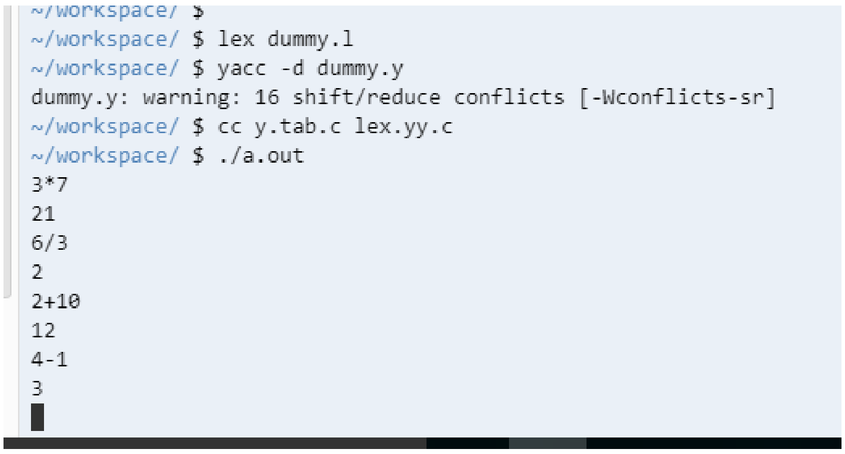
```
%{
        #include <ctype.h>
        #include <stdio.h>
%}

%token DIGIT

%%

line    : expr '\n' { printf("%d\n", $1); return 0;}
        ;
expr    : expr '+' term { $$ = $1 + $3; }
        | term
        ;
term    : term '*' factor { $$ = $1 * $3; }
        | factor
        ;
factor  : '(' expr ')' { $$ = $2; }
        | DIGIT
        ;
%%

yylex()
{
          int c;
          c = getchar();
          if (isdigit(c))
          {
                    yylval = c-'0';
                    return DIGIT;
          }
          return c;
}
```

Output for parser generator using YACC:

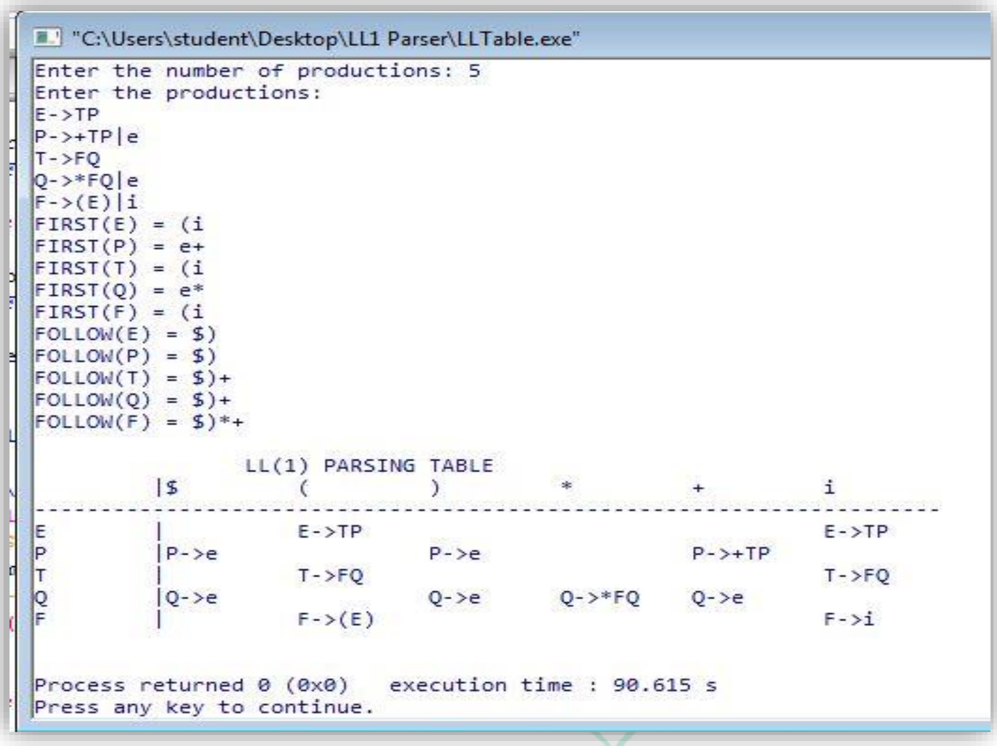


```
~/workspace/ $ yacc parser.y
~/workspace/ $ cc y.tab.c -ly
~/workspace/ $ ./a.out
5+3
8
~/workspace/ $ ./a.out
```

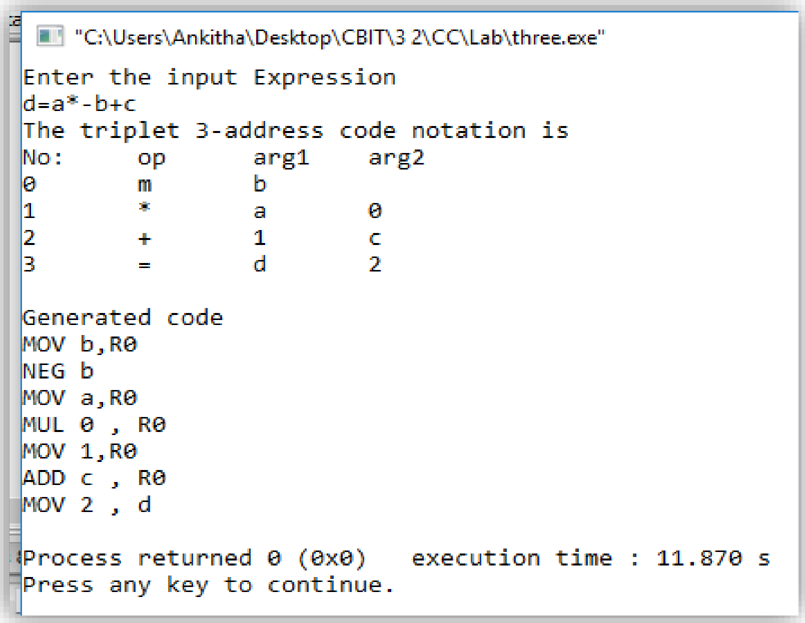
Output for simple calculator using YACC:



```
~/workspace/ $
~/workspace/ $ lex dummy.l
~/workspace/ $ yacc -d dummy.y
dummy.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
~/workspace/ $ cc y.tab.c lex.yy.c
~/workspace/ $ ./a.out
3*7
21
6/3
2
2+10
12
4-1
3
```



```
"C:\Users\student\Desktop\LL1 Parser\LLTable.exe"
Enter the number of productions: 5
Enter the productions:
E->TP
P->+TP|e
T->FQ
Q->*FQ|e
F->(E)|i
FIRST(E) = (i
FIRST(P) = e+
FIRST(T) = (i
FIRST(Q) = e*
FIRST(F) = (i
FOLLOW(E) = $)
FOLLOW(P) = $)
FOLLOW(T) = $)+
FOLLOW(Q) = $)+
FOLLOW(F) = $)*+

                LL(1) PARSING TABLE
         |$         (         )         *         +         i
-----------------------------------------------------------------------
E        |          E->TP                                   E->TP
P        |P->e                P->e                P->+TP
T        |          T->FQ                                   T->FQ
Q        |Q->e                Q->e      Q->*FQ    Q->e
F        |          F->(E)                                  F->i


Process returned 0 (0x0)   execution time : 90.615 s
Press any key to continue.
```



```
"C:\Users\Ankitha\Desktop\CBIT\3 2\CC\Lab\three.exe"
Enter the input Expression
d=a*-b+c
The triplet 3-address code notation is
No:     op      arg1    arg2
0       m       b
1       *       a       0
2       +       1       c
3       =       d       2

Generated code
MOV b,R0
NEG b
MOV a,R0
MUL 0 , R0
MOV 1,R0
ADD c , R0
MOV 2 , d

Process returned 0 (0x0)   execution time : 11.870 s
Press any key to continue.
```