1. Shift Reduce Parser:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int k = 0, z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

void check()
{
   strcpy(ac, "REDUCE TO E");

   // c=length of input string
   for (z = 0; z < c; z++)
      if (stk[z] == 'i' && stk[z + 1] == 'd')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\t%s", stk, a, ac);
         j++;
      }
   for (z = 0; z < c; z++)
      if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         stk[z + 2] = '\0';
         printf("\n$%s\t%s$\t%s", stk, a, ac);
         i = i - 2;
      }
   for (z = 0; z < c; z++)
      if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\t%s", stk, a, ac);
         i = i - 2;
      }
   for (z = 0; z < c; z++)
      if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         stk[z + 1] = '\0';
```

```
        printf("\n$%s\t%s$\t%s", stk, a, ac);
        i = i - 2;
      }
}

int main()
{
   puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");

   // a is input string
   puts("Enter input string: ");
   gets(a);

   c = strlen(a);

   //"SHIFT" is copied to act to be printed
   strcpy(act, "SHIFT->");

   printf("\nStack \t Input \t Action");

   for (k = 0, i = 0; j < c; k++, i++, j++)
   {
      if (a[j] == 'i' && a[j + 1] == 'd')
      {
         stk[i] = a[j];
         stk[i + 1] = a[j + 1];
         stk[i + 2] = '\0';
         a[j] = ' ';
         a[j + 1] = ' ';
         printf("\n$%s\t%s$\t%sid", stk, a, act);
         check();
      }
      else
      {
         stk[i] = a[j];
         stk[i + 1] = '\0';
         a[j] = ' ';
         printf("\n$%s\t%s$\t%ssymbols", stk, a, act);
         check();
      }
   }

   if (stk[0] == 'E' && stk[1] == '\0')
      printf("\n\t\t\tAccept\n");
   else
      printf("\n\t\t\tReject\n");
}
```

```
C:\Users\MAHIN\VII Sem\CD\lab>shiftReduce.exe
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
Enter input string:
E+E*E

Stack    Input   Action
$E       +E*E$  SHIFT->symbols
$E+       E*E$  SHIFT->symbols
$E+E       *E$  SHIFT->symbols
$E         *E$  REDUCE TO E
$E*         E$  SHIFT->symbols
$E*E         $  SHIFT->symbols
$E           $  REDUCE TO E
                       Accept

C:\Users\MAHIN\VII Sem\CD\lab>
```

2. Operator Precedence Parser:

```
/*
The grammar in this program is:
E -> i      / E*E     / E+E     / (E)     / E^E
i for identifier.
E is the start symbol.

*/

#include<stdio.h>
#include<string.h>
#include<stdlib.h>

char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={
        /*stack  +  -  *  /  ^  i  (  )  $ */
        /* + */ '>', '>','<','<','<','<','<','>','>',

        /* - */ '>', '>','<','<','<','<','<','>','>',
```

```
        /*  * */ '>', '>','>','>','<','<','<','>','>',
        /*  / */ '>', '>','>','>','<','<','<','>','>',
        /*  ^ */ '>', '>','>','>','<','<','<','>','>',
        /*  i */ '>', '>','>','>','>','e','e','>','>',
        /*  ( */ '<', '<','<','<','<','<','<','>','e',
        /*  ) */ '>', '>','>','>','>','e','e','>','>',
        /*  $ */ '<', '<','<','<','<','<','<','<','>',

          };

int getindex(char c)
{
switch(c)
   {
   case '+':return 0;
   case '-':return 1;
   case '*':return 2;
   case '/':return 3;
   case '^':return 4;
   case 'i':return 5;
   case '(':return 6;
   case ')':return 7;
   case '$':return 8;
   }
}
```

```
C:\Users\MAHIN\VII Sem\CD\lab>operator_parsing.exe

Enter the string
i+i*i

STACK   INPUT   ACTION
$i      +i*i$   Shift
$E      +i*i$   Reduced: E->i
$E+     i*i$    Shift
$E+i    *i$     Shift
$E+E    *i$     Reduced: E->i
$E      *i$     Reduced: E->E+E
$E*     i$      Shift
$E*i    $       Shift
$E*E    $       Reduced: E->i
$E      $       Reduced: E->E*E
$E$             Shift
$E$             Shift
Accepted;
C:\Users\MAHIN\VII Sem\CD\lab>
```

3. Predictive Parser:

```
#include<iostream>
#include<string>
#include<deque>
using namespace std;
int n,n1,n2;
int getPosition(string arr[], string q, int size)
{
   for(int i=0;i<size;i++)
   {
      if(q == arr[i])
         return i;
   }
   return -1;
}
int main()
{
   string prods[10],first[10],follow[10],nonterms[10],terms[10];
   string pp_table[20][20] = {};
   cout<<"Enter the number of productions : ";
   cin>>n;
   cin.ignore();
   cout<<"Enter the productions"<<endl;
   for(int i=0;i<n;i++)
   {
      getline(cin,prods[i]);
      cout<<"Enter first for "<<prods[i].substr(3)<<" : ";
      getline(cin,first[i]);
   }
   cout<<"Enter the number of Terminals : ";
   cin>>n2;
   cin.ignore();
   cout<<"Enter the Terminals"<<endl;
   for(int i=0;i<n2;i++)
   {
      cin>>terms[i];
   }
   terms[n2] = "$";
   n2++;
   cout<<"Enter the number of Non-Terminals : ";
   cin>>n1;
   cin.ignore();
   for(int i=0;i<n1;i++)
   {
      cout<<"Enter Non-Terminal : ";
      getline(cin,nonterms[i]);
      cout<<"Enter follow of "<<nonterms[i]<<" : ";
```

```
   getline(cin,follow[i]);
}


cout<<endl;
cout<<"Grammar"<<endl;
for(int i=0;i<n;i++)
{
   cout<<prods[i]<<endl;
}


for(int j=0;j<n;j++)
{
   int row = getPosition(nonterms,prods[j].substr(0,1),n1);
   if(prods[j].at(3)!='#')
   {
      for(int i=0;i<first[j].length();i++)
      {
         int col = getPosition(terms,first[j].substr(i,1),n2);
         pp_table[row][col] = prods[j];
      }
   }
   else
   {
      for(int i=0;i<follow[row].length();i++)
      {
         int col = getPosition(terms,follow[row].substr(i,1),n2);
         pp_table[row][col] = prods[j];
      }
   }
}
//Display Table
for(int j=0;j<n2;j++)
   cout<<"\t"<<terms[j];
cout<<endl;
for(int i=0;i<n1;i++)
{
      cout<<nonterms[i]<<"\t";
      //Display Table
      for(int j=0;j<n2;j++)
      {
         cout<<pp_table[i][j]<<"\t";
      }
      cout<<endl;
}
//Parsing String
```

```
char c;
do{
string ip;
deque<string> pp_stack;
pp_stack.push_front("$");
pp_stack.push_front(prods[0].substr(0,1));
cout<<"Enter the string to be parsed : ";
getline(cin,ip);
ip.push_back('$');
cout<<"Stack\tInput\tAction"<<endl;
while(true)
{
   for(int i=0;i<pp_stack.size();i++)
      cout<<pp_stack[i];
   cout<<"\t"<<ip<<"\t";
   int row1 = getPosition(nonterms,pp_stack.front(),n1);
   int row2 = getPosition(terms,pp_stack.front(),n2);
   int column = getPosition(terms,ip.substr(0,1),n2);
   if(row1 != -1 && column != -1)
   {
      string p = pp_table[row1][column];
      if(p.empty())
      {
         cout<<endl<<"String cannot be Parsed."<<endl;
         break;
      }
      pp_stack.pop_front();
      if(p[3] != '#')
      {
         for(int x=p.size()-1;x>2;x--)
         {
            pp_stack.push_front(p.substr(x,1));
         }
      }
      cout<<p;
   }
   else
   {
      if(ip.substr(0,1) == pp_stack.front())
      {
         if(pp_stack.front() == "$")
         {
            cout<<endl<<"String Parsed."<<endl;
            break;
         }
         cout<<"Match "<<ip[0];
         pp_stack.pop_front();
```

```
            ip = ip.substr(1);
          }
          else
          {
            cout<<endl<<"String cannot be Parsed."<<endl;
            break;
          }
        }
        cout<<endl;
    }
    cout<<"Continue?(Y/N) ";
    cin>>c;
    cin.ignore();
    }while(c=='y' || c=='Y');
    return 0;
}
```

```
C:\Users\MAHIN\VII Sem\CD\lab>predictive_parsing.exe
Enter the number of productions : 5
Enter the productions
S->aXYb
Enter first for aXYb : a
X->c
Enter first for c : c
X->#
Enter first for # : #
Y->d
Enter first for d : d
Y->#
Enter first for # : #
Enter the number of Terminals : 4
Enter the Terminals
a
b
c
d
Enter the number of Non-Terminals : 3
Enter Non-Terminal : S
Enter follow of S : $
Enter Non-Terminal : X
Enter follow of X : bd
Enter Non-Terminal : Y
Enter follow of Y : b

Grammar
S->aXYb
X->c
X->#
Y->d
Y->#
        a       b       c       d       $
S       S->aXYb
X               X->#    X->c    X->#
Y               Y->#            Y->d
Enter the string to be parsed : acdb
Stack   Input   Action
S$      acdb$   S->aXYb
aXYb$   acdb$   Match a
XYb$    cdb$    X->c
cYb$    cdb$    Match c
Yb$     db$     Y->d
db$     db$     Match d
b$      b$      Match b
$       $
String Parsed.
```

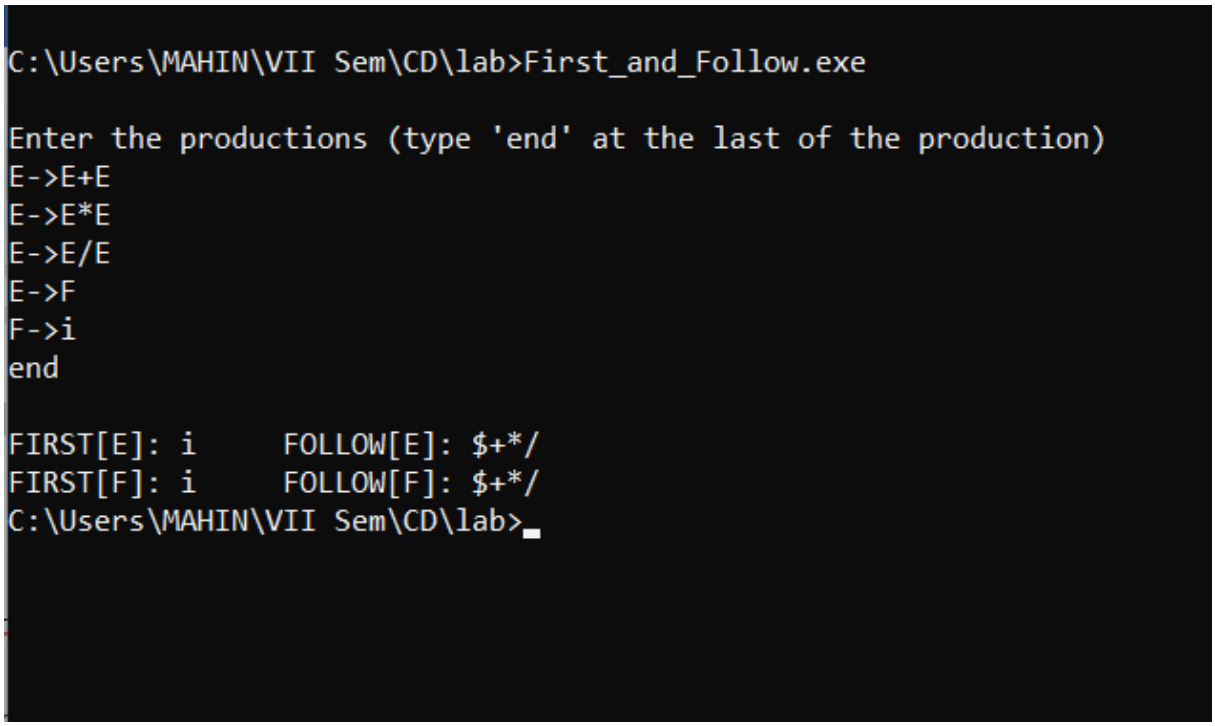
4. First And Follow:

```
#include <bits/stdc++.h>
#define max 20
using namespace std;
char prod[max][10], ter[10], nt[10], first[10][10], follow[10][10];
int eps[10],c=0;
int findpos(char ch)
{
   int n;
   for (n = 0; nt[n] != '\0'; n++)
      if (nt[n] == ch)
         break;
   if (nt[n] == '\0')
      return 1;
   return n;
}
int IsCap(char c)
{
   return (c >= 'A' && c <= 'Z') ? 1 : 0;
}
void add(char *arr, char c)
{
   int i, flag = 0;
   for (i = 0; arr[i] != '\0'; i++)
      if (arr[i] == c)
      {
         flag = 1;
         break;
      }
   if (flag != 1)
      arr[strlen(arr)] = c;
}
void addarr(char *s1, char *s2)
{
   int i, j, flag = 99;
   for (i = 0; s2[i] != '\0'; i++)
   {
      flag = 0;
      for (j = 0;; j++)
      {
         if (s2[i] == s1[j])
         {
            flag = 1;
            break;
         }
         if (j == strlen(s1) && flag != 1)
         {
```

```
            s1[strlen(s1)] = s2[i];
            break;
         }
      }
   }
}
void addprod(char *s)
{
   int i;
   prod[c][0] = s[0];
   for (i = 3; s[i] != '\0'; i++)
   {
      if (!IsCap(s[i]))
         add(ter, s[i]);
      prod[c][i - 2] = s[i];
   }
   prod[c][i - 2] = '\0';
   add(nt, s[0]);
   c++;
}
void findfirst()
{
   int i, j, n, k, e, n1;
   for (i = 0; i < c; i++)
   {
      for (j = 0; j < c; j++)
      {
         n = findpos(prod[j][0]);
         if (prod[j][1] == (char)238)
            eps[n] = 1;
         else
         {
            for (k = 1, e = 1; prod[j][k] != '\0' && e == 1; k++)
            {
               if (!IsCap(prod[j][k]))
               {
                  e = 0;
                  add(first[n], prod[j][k]);
               }
               else
               {
                  n1 = findpos(prod[j][k]);
                  addarr(first[n], first[n1]);
                  if (eps[n1] == 0)
                     e = 0;
               }
            }
```

```
            if (e == 1)
               eps[n] = 1;
         }
      }
   }
}
void findfollow()
{
   int i, j, k, n, e, n1;
   n = findpos(prod[0][0]);
   add(follow[n], '$');
   for (i = 0; i < c; i++)
   {
      for (j = 0; j < c; j++)
      {
         for (k = strlen(prod[j]) - 1; k > 0; k--)
         {
            if (IsCap(prod[j][k]))
            {
               n = findpos(prod[j][k]);
               if (prod[j][k + 1] == '\0') // A -> aB
               {
                  n1 = findpos(prod[j][0]);
                  addarr(follow[n], follow[n1]);
               }
               if (IsCap(prod[j][k + 1])) // A -> aBb
               {
                  n1 = findpos(prod[j][k + 1]);
                  addarr(follow[n], first[n1]);
                  if (eps[n1] == 1)
                  {
                     n1 = findpos(prod[j][0]);
                     addarr(follow[n], follow[n1]);
                  }
               }
               else if (prod[j][k + 1] != '\0')
                  add(follow[n], prod[j][k + 1]);
            }
         }
      }
   }
}
int main()
{
   char s[max], i;
   cout << "\nEnter the productions (type 'end' at the last of the production)\n";
   cin >> s;
```

```
  while (strcmp("end", s))
  {
    addprod(s);
    cin >> s;
  }
  findfirst();
  findfollow();
  for (i = 0; i < strlen(nt); i++)
  {
    cout << "\nFIRST[" << nt[i] << "]: " << first[i];
    if (eps[i] == 1)
      cout << (char)238 << "\t";
    else
      cout << "\t";
    cout << "FOLLOW[" << nt[i] << "]: " << follow[i];
  }
  return 0;
}
```



5. Leading and Trailing:

```
#include<bits/stdc++.h>
using namespace std;
#include <cstring>
int nt, t, top = 0;
char s[50], NT[10], T[10], st[50], l[10][10], tr[50][50];
int searchnt(char a)
{
  int count = -1, i;
  for (i = 0; i < nt; i++)
```

```
    {
        if (NT[i] == a)
            return i;
    }
    return count;
}
int searchter(char a)
{
    int count = -1, i;
    for (i = 0; i < t; i++)
    {
        if (T[i] == a)
            return i;
    }
    return count;
}
void push(char a)
{
    s[top] = a;
    top++;
}
char pop()
{
    top--;
    return s[top];
}
void installl(int a, int b)
{
    if (l[a][b] == 'f')
    {
        l[a][b] = 't';
        push(T[b]);
        push(NT[a]);
    }
}
void installt(int a, int b)
{
    if (tr[a][b] == 'f')
    {
        tr[a][b] = 't';
        push(T[b]);
        push(NT[a]);
    }
}
int main()
{
    int i, s, k, j, n;
```

```
char pr[30][30], b, c;
cout<< "Enter the no of productions:";
cin>> n;
cout << "Enter the productions one by one\n";
for (i = 0; i < n; i++)
   cin >> pr[i];
nt = 0;
t = 0;
for (i = 0; i < n; i++)
{
   if ((searchnt(pr[i][0])) == -1)
      NT[nt++] = pr[i][0];
}
for (i = 0; i < n; i++)
{
   for (j = 3; j < strlen(pr[i]); j++)
   {
      if (searchnt(pr[i][j]) == -1)
      {
         if (searchter(pr[i][j]) == -1)
            T[t++] = pr[i][j];
      }
   }
}
for (i = 0; i < nt; i++)
{
   for (j = 0; j < t; j++)
      l[i][j] = 'f';
}
for (i = 0; i < nt; i++)
{
   for (j = 0; j < t; j++)

      tr[i][j] = 'f';
}
for (i = 0; i < nt; i++)
{
   for (j = 0; j < n; j++)
   {
      if (NT[(searchnt(pr[j][0]))] == NT[i])
      {
         if (searchter(pr[j][3]) != -1)
            installl(searchnt(pr[j][0]), searchter(pr[j][3]));
         else
         {
            for (k = 3; k < strlen(pr[j]); k++)
            {
```

```
                if (searchnt(pr[j][k]) == -1)
                {
                   installl(searchnt(pr[j][0]), searchter(pr[j][k]));
                   break;
                }
             }
          }
       }
    }
}
while (top != 0)
{
    b = pop();
    c = pop();
    for (s = 0; s < n; s++)
    {
       if (pr[s][3] == b)
          installl(searchnt(pr[s][0]), searchter(c));
    }
}
for (i = 0; i < nt; i++)
{
    cout << "Leading[" << NT[i] << "]"
        << "\t{";
    for (j = 0; j < t; j++)
    {
       if (l[i][j] == 't')
          cout << T[j] << ",";
    }
    cout << "}\n";
}
top = 0;
for (i = 0; i < nt; i++)
{
    for (j = 0; j < n; j++)
    {
       if (NT[searchnt(pr[j][0])] == NT[i])
       {
          if (searchter(pr[j][strlen(pr[j]) - 1]) != -1)
             installt(searchnt(pr[j][0]), searchter(pr[j][strlen(pr[j]) - 1]));
          else
          {
             for (k = (strlen(pr[j]) - 1); k >= 3; k--)
             {
                if (searchnt(pr[j][k]) == -1)
                {
                   installt(searchnt(pr[j][0]), searchter(pr[j][k]));
```

```
                    break;
                }
            }
        }
    }
  }
  while (top != 0)
  {
    b = pop();
    c = pop();
    for (s = 0; s < n; s++)
    {
      if (pr[s][3] == b)
        installt(searchnt(pr[s][0]), searchter(c));
    }
  }
  for (i = 0; i < nt; i++)
  {
    cout << "Trailing[" << NT[i] << "]"
      << "\t{";
    for (j = 0; j < t; j++)
    {
      if (tr[i][j] == 't')
        cout << T[j] << ",";
    }
    cout << "}\n";
  }
  return 0;
}
```

```
C:\Users\MAHIN\VII Sem\CD\lab>Leading_and_Trailing.exe
Enter the no of productions:5
Enter the productions one by one
E->E+E
E->E*E
E->E/E
E->F
F->i
Leading[E]      {+,*,/,i,}
Leading[F]      {i,}
Trailing[E]     {+,*,/,i,}
Trailing[F]     {i,}

C:\Users\MAHIN\VII Sem\CD\lab>
```

6. LR(0):

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

typedef map<char, vector<string> > AugmentedGrammar;
typedef map<string, int> GotoMap; // <aps productions to LR(0) item ids

// structy for representing an augmented grammar production
struct AugmentedProduction
{
   char lhs;   // left hand side of production
   string rhs; // right hand side of production

   AugmentedProduction() {}
   AugmentedProduction(char _lhs, string _rhs) : lhs(_lhs), rhs(_rhs) {}
};

// Class for representing LR(0) items.
class LR0Item
{
private:
   // list of productions
   vector<AugmentedProduction*> productions;

public:
   // list of out-edges
   map<char, int> gotos;

   // constructor
   LR0Item() {}
   // destructor
   ~LR0Item() {}

   // add production
   void Push(AugmentedProduction *p) { productions.push_back(p); }
   // return the number of productions
   int Size() { return int(productions.size()); }

   // return whether or not this item contains the production prodStr
   bool Contains (string production) {
```

```
      for (auto it = productions.begin(); it != productions.end(); it++) {
         string existing = string(&(*it)->lhs, 1) + "->" + (*it)->rhs;
         if (strcmp(production.c_str(), existing.c_str()) == 0) {
            return true;
         }
      }
      return false;
   }

   // overloaded index operator; access pointer to production.
   AugmentedProduction* operator[](const int index) {
      return productions[index];
   }
};

/* void add_closure
 * If 'next' is the current input symbol and next is nonterminal, then the set
 * of LR(0) items reachable from here on next includes all LR(0) items reachable
 * from here on FIRST(next). Add all grammar productions with a lhs of next */
void
add_closure(char lookahead, LR0Item& item, AugmentedGrammar& grammar)
{
   // only continue if lookahead is a non-terminal
   if (!isupper(lookahead)) return;

   string lhs = string(&lookahead, 1);
   // iterate over each grammar production beginning with p->rhs[next]
   // to see if that production has already been included in this item.
   for (int i = 0; i<grammar[lookahead].size(); i++) {
      string rhs = "." + grammar[lookahead][i];
      // if the grammar production for the next input symbol does not yet
      // exist for this item, add it to the item's set of productions
      if (!item.Contains( lhs + "->" + rhs )) {
         item.Push(new AugmentedProduction(lookahead, rhs));
      }
   }
}

// produce the graph of LR(0) items from the given augmented grammar
void
get_LR0_items(vector<LR0Item>& lr0items, AugmentedGrammar& grammar, int& itemid,
GotoMap& globalGoto)
{
   printf("I%d:\n", itemid);

   // ensure that the current item contains te full closure of it's productions
   for (int i = 0; i<lr0items[itemid].Size(); i++) {
```

```
      string rhs = lr0items[itemid][i]->rhs;
      char lookahead = rhs[rhs.find('.')+1];
      add_closure(lookahead, lr0items[itemid], grammar);
    }

    int nextpos;
    char lookahead, lhs;
    string rhs;
    AugmentedProduction *prod;

    // iterate over each production in this LR(0) item
    for (int i = 0; i<lr0items[itemid].Size(); i++) {
      // get the current production
      lhs = lr0items[itemid][i]->lhs;
      rhs = lr0items[itemid][i]->rhs;
      string production = string(&lhs,1) + "->" + rhs;

      // get lookahead if one exists
      lookahead = rhs[rhs.find('.')+1];
      if (lookahead == '\0') {
        printf("\t%-20s\n", &production[0]);
        continue;
      }

      // if there is no goto defined for the current input symbol from this
      // item, assign one.
      if (lr0items[itemid].gotos.find(lookahead) == lr0items[itemid].gotos.end()) {
        // that one instead of creating a new one
        // if there is a global goto defined for the entire production, use
        if (globalGoto.find(production) == globalGoto.end()) {
          lr0items.push_back(LR0Item()); // create new state (item)
          // new right-hand-side is identical with '.' moved one space to the right
          string newRhs = rhs;
          int atpos = newRhs.find('.');
          swap(newRhs[atpos], newRhs[atpos+1]);
          // add item and update gotos
          lr0items.back().Push(new AugmentedProduction(lhs, newRhs));
          lr0items[itemid].gotos[lookahead] = lr0items.size()-1;
          globalGoto[production] = lr0items.size()-1;
        } else {
          // use existing global item
          lr0items[itemid].gotos[lookahead] = globalGoto[production];
        }
        printf("\t%-20s goto(%c)=I%d\n", &production[0], lookahead,
globalGoto[production]);
      } else {
        // there is a goto defined, add the current production to it
```

```
            // move . one space to right for new rhs
            int at = rhs.find('.');
            swap(rhs[at], rhs[at+1]);
            // add production to next item if it doesn't already contain it
            int nextItem = lr0items[itemid].gotos[lookahead];
            if (!lr0items[nextItem].Contains(string(&lhs, 1) + "->" + rhs)) {
               lr0items[nextItem].Push(new AugmentedProduction(lhs, rhs));
            }
            swap(rhs[at], rhs[at+1]);
            printf("\t%-20s\n", &production[0]);
         }
      }
   }

   /**
    * void load_grammar
    * scan and load the grammar from stdin while setting first LR(0) item */
   void load_grammar(AugmentedGrammar& grammar, vector<LR0Item>& lr0items)
   {
      string production;
      string lhs, rhs;
      string delim = "->";

      getline(cin, lhs); // scan start production
      grammar['\''].push_back(lhs);
      lr0items[0].Push(new AugmentedProduction('\'', "." + lhs));
      printf("'->%s\n", lhs.c_str());

      while(1) {
         getline(cin, production);
         if (production.length() < 1) return;

         auto pos = production.find(delim);
         if(pos!=string::npos){
            lhs = production.substr(0,pos);
            rhs = production.substr(pos+delim.length(),std::string::npos);
         }

         grammar[lhs[0]].push_back(rhs);
         printf("%s->%s\n", lhs.c_str(), rhs.c_str());
         lr0items[0].Push(new AugmentedProduction(lhs[0], "." + rhs));
      }
   }

   // main
   int main() {
      int itemid = -1; // counter for the number of LR(0) items
```

```
    AugmentedGrammar grammar;
    vector<LR0Item> lr0items = { LR0Item() }; // push start state
    GotoMap globalGoto;

    printf("Augmented Grammar\n");
    printf("-----------------\n");
    load_grammar(grammar, lr0items);
    printf("\n");

    printf("Sets of LR(0) Items\n");
    printf("-------------------\n");
    while (++itemid < int(lr0items.size())) {
        get_LR0_items(lr0items, grammar, itemid, globalGoto);
    }
    printf("\n");
    return 0;
}
```

```
C:\Users\MAHIN\VII Sem\CD\lab>LR0.exe
Augmented Grammar
-----------------
S->CC
'->S->CC
C->cC
C->cC
C->d
C->d


Sets of LR(0) Items
-------------------
I0:
        '->.S->CC               goto(S)=I1
        C->.cC                  goto(c)=I2
        C->.d                   goto(d)=I3
I1:
        '->S.->CC               goto(-)=I4
I2:
        C->c.C                  goto(C)=I5
        C->.cC                  goto(c)=I2
        C->.d                   goto(d)=I3
I3:
        C->d.
I4:
        '->S-.>CC               goto(>)=I6
I5:
        C->cC.
I6:
        '->S->.CC               goto(C)=I7
        C->.cC                  goto(c)=I2
        C->.d                   goto(d)=I3
I7:
        '->S->C.C               goto(C)=I8
        C->.cC                  goto(c)=I2
        C->.d                   goto(d)=I3
I8:
        '->S->CC.
```