/* B8: Write a program to design SLR parsing. */

```
#include <bits/stdc++.h>
#include <string>
using namespace std;

/*
     First SLR table is created using the given grammer.
     Then given a string finds whether it can be produced by the
grammer.
*/

// p is productions, nt is non terminals, term is terminals
vector< pair<char,string> > p,sets[20];
int z,k;
vector<char> nt,term;
map<char,int> m,mt;
int table[20][20]; vector<int> tt[20][20];
queue<int> Q1;
vector<char> first[10],follow[10];
vector<char> ep;
int visited[10];

// Check for non terminal characters
bool is_nonterm(char c)
{
     if(c>='A' && c<='Z') { return 1; }
     else { return 0; }
}

// Create SLR sets.
void make_set(pair<char,string> b)
{
     queue<pair<char,string> > Q;
     pair<char,string> x;
     if(find(sets[z].begin(),sets[z].end(),b)==sets[z].end())
     {
          sets[z].push_back(b); Q.push(b);
     }

     while(!Q.empty())
     {
          x = Q.front(); Q.pop();
          int pos,f=1;

          // Find the position of '.'
          for(int i=0;i<x.second.size();i++) { if(x.second[i]=='.') {
pos=i; break; } }

          // If '.' is at the end then we are done with the
producion.
```

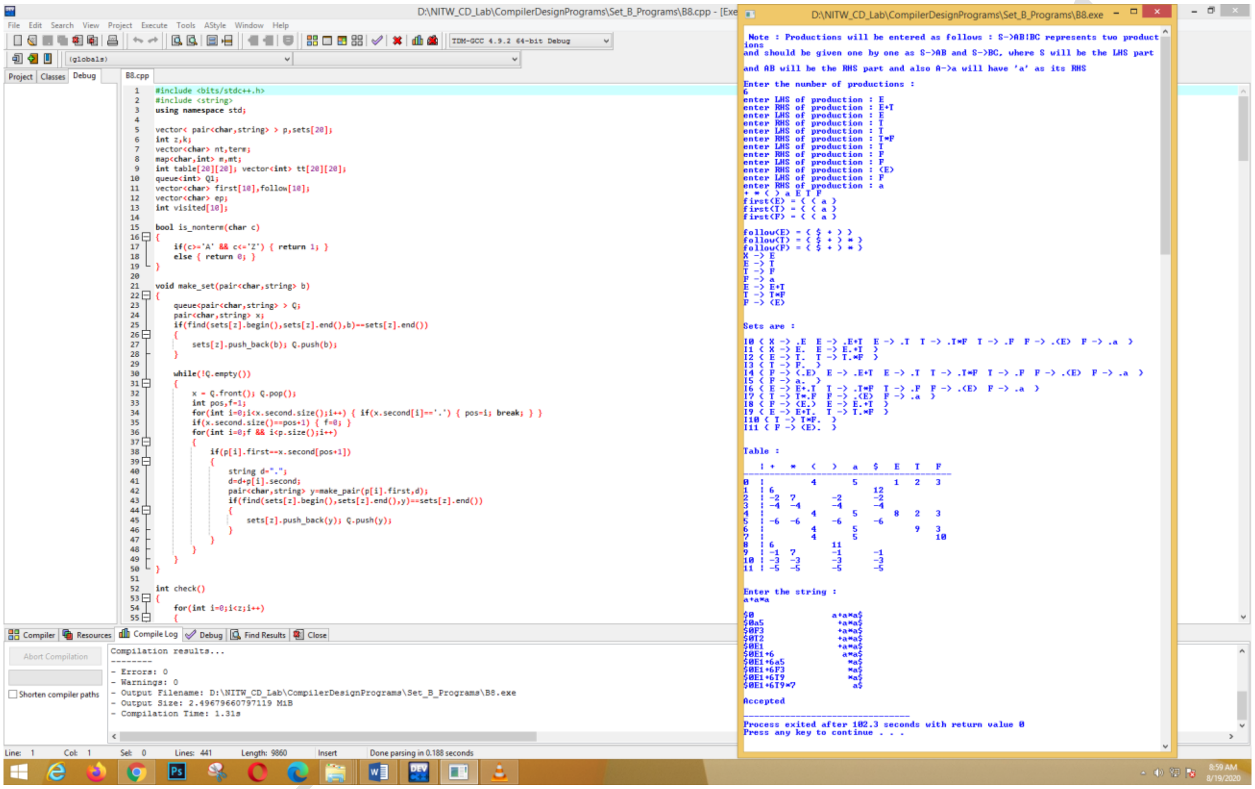
```
            if(x.second.size()==pos+1) { f=0; }
            for(int i=0;f && i<p.size();i++)
            {
                  // If we find a non-terminal in the rhs of the current
set, add it to the queue.
                  if(p[i].first==x.second[pos+1])
                  {
                        string d=".";
                        d=d+p[i].second;
                        pair<char,string> y=make_pair(p[i].first,d);

      if(find(sets[z].begin(),sets[z].end(),y)==sets[z].end())
                        {
                              sets[z].push_back(y); Q.push(y);
                        }
                  }
            }
      }
}

// Compare sets.
int check()
{
      for(int i=0;i<z;i++)
      {
            if(sets[i]==sets[z]) { return i; }
      }
      return -1;
}

// Using goto create the slr parsing table
void goto_fun(char g,int ind)
{
      for(int j=0;j<sets[ind].size();j++)
      {
            int found=0,kkk=0;
            // Find if the required character is after a '.'
            for(int h=0;h<sets[ind][j].second.size();h++)
            {
                  if(sets[ind][j].second[h]=='.' &&
h+1<sets[ind][j].second.size() && sets[ind][j].second[h+1]==g)
                  { found=h; kkk=1; break; }
            }
            // if the required character is after a '.', create sets
            if(kkk)
            {
                  string xx="";
                  for(int h=0;h<found;h++) {
xx=xx+sets[ind][j].second[h]; }
                  xx=xx+sets[ind][j].second[found+1]; xx=xx+".";
                  for(int h=found+2;h<sets[ind][j].second.size();h++) {
xx=xx+sets[ind][j].second[h]; }
```

```
                  make_set(make_pair(sets[ind][j].first,xx));
            }
      }
      if(sets[z].size()==0) { return; }
      int same;
      same=check();
      if(same==-1)
      {
            table[ind][m[g]]=z;
            if(
find(tt[ind][m[g]].begin(),tt[ind][m[g]].end(),z)==tt[ind][m[g]].end()
) { tt[ind][m[g]].push_back(z); }
            Q1.push(z); z++;
      }
      else
      {
            table[ind][m[g]]=same; sets[z].clear();
            if(
find(tt[ind][m[g]].begin(),tt[ind][m[g]].end(),same)==tt[ind][m[g]].en
d() ) { tt[ind][m[g]].push_back(same); }
      }
}

// Check for epsilon production and mark it.
void mark_epsilon()
{
      for(int i=0;i<p.size();i++)
      {
            if(p[i].second.size()==0) { ep.push_back(p[i].first); }
      }
}

// Traverse all the productions using dfs and find the first.
void first_util(char lhs,vector<char> &v)
{
      if(visited[mt[lhs]]==1) { return; }
      visited[mt[lhs]]=1;
      for(int i=0;i<p.size();i++)
      {
            if(lhs==p[i].first && p[i].second.size()>0)
            {
                  int max_l=p[i].second.size(),j=0;
                  while(j<max_l)
                  {     // We found a terminal character, so if it is not
already in first of lhs, add it.
                        if(!is_nonterm(p[i].second[j]))
                        {

      if(find(v.begin(),v.end(),p[i].second[j])==v.end()) {
v.push_back(p[i].second[j]); }
                              break;
                        }
                        else
                        {
                              first_util(p[i].second[j],v);
                              if(
find(ep.begin(),ep.end(),p[i].second[j])==ep.end() ) { break; }
                              j++;
                        }
                  }
                  if(j==max_l) { ep.push_back(lhs); }
            }
      }
}

// For each non terminal find its first.
void find_first()
{
      for(int i=0;i<nt.size();i++)
      {
            memset(visited,0,10*sizeof(int));
            char c=nt[i];
            first_util(c,first[mt[nt[i]]]);
      }
}

// Traverse all the productions using dfs and find the follow.
void follow_util(char lhs,vector<char> &v)
{
      if(visited[mt[lhs]]==1) { return; }
      visited[mt[lhs]]=1;

      // Follow of the added production in augmented grammer is always
epsilon.
      if(lhs==p[0].first) { if(find(v.begin(),v.end(),'$')==v.end()) {
v.push_back('$'); }  }

      // Iterate through all productions
      for(int i=0;i<p.size();i++)
      {
            int max_l=p[i].second.size();
            for(int j=0;j<p[i].second.size();j++)
            {
                  // If we find the required character in the rhs of
production.
                  if(p[i].second[j]==lhs)
                  {
                        j++;
                        while(j<max_l)
                        {
                              // We found a terminal after the given non
terminal which means it is the follow.
                              if(!is_nonterm(p[i].second[j]))
                              {
```

```
      if(find(v.begin(),v.end(),p[i].second[j])==v.end()) {
v.push_back(p[i].second[j]); }
                                    break;
                              }
                              else
                              {
                                    // Else find the first of the non
terminal which will the follow of the required non terminal.
                                    int index=mt[p[i].second[j]];
                                    for(int l=0;l<first[index].size();l++)
                                    {

      if(find(v.begin(),v.end(),first[index][l])==v.end()) {
v.push_back(first[index][l]); }
                                    }
                                    if(
find(ep.begin(),ep.end(),p[i].second[j])==ep.end() ) { break; }
                                    j++;
                              }
                        }
                        if(j==max_l) { follow_util(p[i].first,v); }
                  }
            }
      }
}

// For each non terminal find its follow.
void find_follow()
{
      for(int i=0;i<nt.size();i++)
      {
            memset(visited,0,10*sizeof(int));
            char c=nt[i];
            follow_util(c,follow[mt[nt[i]]]);
      }
}

int find_prod(pair<char,string> x)
{
      for(int i=0;i<p.size();i++)
      {
            if(p[i]==x) { return (i+1); }
      }
      return -1;
}

void print_stack(stack<char> s1,stack<int> states)
{
      string h=""; int a[20],i=0;
      while(!s1.empty()) { h=h+s1.top(); s1.pop(); }
      while(!states.empty()) { a[i++]=states.top(); states.pop(); }
```

```
    for(int j=h.size()-1;j>=0;j--) { cout<<a[j+1]<<h[j]; }
    cout<<a[0];
}

// Given a string check if it can be parsed by the given slr parser
using slr table.
void parsing()
{
    string input;
    cout<<"\n\nEnter the string : \n";
    cin>>input;
    cout<<"\n";
    stack<char> s1; stack<int> states;
    int ptr=0;
    states.push(0);
    while(ptr<input.size())
    {
        cout<<"$";
        print_stack(s1,states);
        for(int i=0;i<20-2*s1.size()-input.size()+ptr;i++) {
cout<<" "; }
        for(int i=ptr;i<input.size();i++) { cout<<input[i]; }
        cout<<"$\n";
        int x=states.top(),y=m[input[ptr]];
        if(table[x][y]>0)
        {
            s1.push(input[ptr]); ptr++;
            states.push(table[x][y]);
        }
        else if(table[x][y]<0)
        {
            int pn=(-1)*table[x][y];
            pn--;
            if(s1.size()<p[pn].second.size() &&
states.size()<p[pn].second.size()) { cout<<"Error1\n"; exit(0); }
            for(int i=0;i<p[pn].second.size();i++)
            {
                s1.pop(); states.pop();
            }
            s1.push(p[pn].first);
            states.push(table[states.top()][m[s1.top()]]);
        }
        else if(table[x][y]==0) { cout<<"Error2\n"; exit(0); }
    }
    cout<<"\nAccepted\n";
}

// Prints the created slr table
void print_tt()
{
    cout<<"\n\nTable : \n\n   | ";
    for(int i=0;i<term.size();i++) { cout<<term[i]<<"     "; }
```

```
      for(int i=0;i<nt.size();i++) { cout<<nt[i]<<"     "; }
      cout<<"\n";
      cout<<"------------------------------------------\n";
      for(int i=0;i<z;i++)
      {
            cout<<i<<" ";
            if(i<10) { cout<<" "; }
            cout<<"| ";
            for(int j=0;j<k;j++)
            {
                  for(int e=0;e<tt[i][j].size();e++)
                  {
                        if(e>0) { cout<<","; } cout<<tt[i][j][e];
                  }
                  if(tt[i][j].size()==0) { cout<<"  "; }
                  cout<<"  ";
            }
            cout<<"\n";
      }
}

int main()
{
      // Input all the productions.
      int n;
      char c;
      string s;
      cout<<"\n Note : Productions will be entered as follows : S-
>AB|BC represents two productions \n";
    cout<<"and should be given one by one as S->AB and S->BC, where S
will be the LHS part \n";
    cout<<"and AB will be the RHS part and also A->a will have 'a' as
its RHS \n\n";
      cout<<"Enter the number of productions : \n";
      cin>>n;
      for(int i=0;i<n;i++)
      {
            cout<<"enter LHS of production : ";
            cin>>c;
            cout<<"enter RHS of production : ";
            cin>>s;
            if(s=="%") { s=""; }
            p.push_back(make_pair(c,s));
      }

      // Finding terminals and non-terminals
      k=0;
      for(int i=0;i<p.size();i++)
      {
            int x=1;
            // If it is the first time seeing lhs store it in nt.
            if(find(nt.begin(),nt.end(),p[i].first)==nt.end())
            {
                  nt.push_back(p[i].first);
            }
            // In the rhs check if the character is non term and also
if it is the first time seeing it, store in term.
            for(int j=0;j<p[i].second.size();j++)
            {
                  if(!is_nonterm(p[i].second[j]) &&
find(term.begin(),term.end(),p[i].second[j])==term.end())
                  {
                        term.push_back(p[i].second[j]);
m[p[i].second[j]]=k++; cout<<p[i].second[j]<<" ";
                  }
            }
      }
      term.push_back('$');
      m['$']=k++;

      for(int i=0;i<nt.size();i++) { m[nt[i]]=k++; mt[nt[i]]=i;
cout<<nt[i]<<" "; }
      cout<<"\n";

      mark_epsilon();
      // Find first and print it.
      find_first();
      for(int i=0;i<nt.size();i++)
      {
            cout<<"first("<<nt[i]<<") = { ";
            for(int j=0;j<first[mt[nt[i]]].size();j++)
            {
                  cout<<first[mt[nt[i]]][j]<<" ";
            }
            cout<<"}\n";
      }

      cout<<endl;

      // Find follow and print it
      find_follow();
      for(int i=0;i<nt.size();i++)
      {
            cout<<"follow("<<nt[i]<<") = { ";
            for(int j=0;j<follow[mt[nt[i]]].size();j++)
            {
                  cout<<follow[mt[nt[i]]][j]<<" ";
            }
            cout<<"}\n";
      }
      /////////

      // Creating sets and goto
      string h=".";
      h=h+p[0].first;
      z=0;

      // X is the added non-terminal in our augmented grammer.
      make_set(make_pair('X',h));
      z++;

      memset(table,0,sizeof(int)*400);
      Q1.push(0);

      // Find goto
      while(!Q1.empty())
      {
            int ind=Q1.front(); Q1.pop();
            char g;
            for(int i=0;i<nt.size();i++)
            {
                  g=nt[i];
                  goto_fun(g,ind);
            }

            for(int i=0;i<term.size();i++)
            {
                  g=term[i];
                  goto_fun(g,ind);
            }
      }
      /////////

      // Finding reduction entries
      for(int i=0;i<z;i++)
      {
            for(int j=0;j<sets[i].size();j++)
            {
                  int last_i = sets[i][j].second.size();
                  if(sets[i][j].second[last_i-1]=='.')
                  {
                        string rhs=sets[i][j].second.substr(0,last_i-1);
                        cout<<sets[i][j].first<<" -> "<<rhs<<"\n";
                        int prod_num =
find_prod(make_pair(sets[i][j].first,rhs));
                        if(prod_num<0) { table[i][m['$']]=z; continue; }
                        prod_num=(-1)*prod_num;
                        int index = mt[sets[i][j].first];
                        for(int l=0;l<follow[index].size();l++)
                        {
                              table[i][m[follow[index][l]]] = prod_num;
                              int pos=m[follow[index][l]];
                              if(
find(tt[i][pos].begin(),tt[i][pos].end(),prod_num)==tt[i][pos].end() )
{ tt[i][pos].push_back(prod_num); }
                        }
            }
        }
    }
    ///////

    // Printing Table and Sets of items...
    cout<<"\n\nSets are :\n\n";
    for(int j=0;j<z;j++)
    {
        cout<<"I"<<j<<" { ";
        for(int i=0;i<sets[j].size();i++)
        {
            cout<<sets[j][i].first<<" -> "<<sets[j][i].second<<"
";
        }
        cout<<"}\n";
    }
    int multiple_ent=0;
    for(int i=0;i<z;i++)
    {
        for(int j=0;j<k;j++)
        {
            if(tt[i][j].size()>1) { multiple_ent=1; break; }
        }
    }

    if(multiple_ent)
    {
        print_tt(); exit(0);
    }
    else
    {
        cout<<"\n\nTable : \n\n    | ";
        for(int i=0;i<term.size();i++) { cout<<term[i]<<"    "; }
        for(int i=0;i<nt.size();i++) { cout<<nt[i]<<"    "; }
        cout<<"\n";
        cout<<"----------------------------------------\n";
        for(int i=0;i<z;i++)
        {
            cout<<i<<" ";
            if(i<10) { cout<<" "; }
            cout<<"| ";
            for(int j=0;j<k;j++)
            {
                if(table[i][j]==0) { cout<<"  "; }
                else { cout<<table[i][j]<<" "; }
                if(table[i][j]<10 && table[i][j]>=0) { cout<<" ";
}
                cout<<" ";
            }
            cout<<"\n";
        }
        }
        ////////
        parsing();
        return 0;
}
```


D:\NITW_CD_Lab\CompilerDesignPrograms\Set_B_Programs\B8.cpp - [Executing]
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Debug
(globals)
Project Classes Debug
B8.cpp
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Line: 1 Col: 1 Sel: 0 Lines: 441 Length: 9860 Insert Done parsing in 0.188 seconds
D:\NITW_CD_Lab\CompilerDesignPrograms\Set_B_Programs\B8.exe

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

vector< pair<char,string> > p,sets[20];
int z,k;
vector<char> nt,term;
map<char,int> m,mt;
int table[20][20]; vector<int> tt[20][20];
queue<int> Q1;
vector<char> first[10],follow[10];
vector<char> ep;
int visited[10];

bool is_nonterm(char c)
{
    if(c>='A' && c<='Z') { return 1; }
    else { return 0; }
}

void make_set(pair<char,string> b)
{
    queue<pair<char,string> > Q;
    pair<char,string> x;
    if(find(sets[z].begin(),sets[z].end(),b)==sets[z].end())
    {
        sets[z].push_back(b); Q.push(b);
    }

    while(!Q.empty())
    {
        x = Q.front(); Q.pop();
        int pos,f=1;
        for(int i=0;i<x.second.size();i++) { if(x.second[i]=='.') { pos=i; break; } }
        if(x.second.size()==pos+1) { f=0; }
        for(int i=0;f && i<p.size();i++)
        {
            if(p[i].first==x.second[pos+1])
            {
                string d=".";
                d=d+p[i].second;
                pair<char,string> y=make_pair(p[i].first,d);
                if(find(sets[z].begin(),sets[z].end(),y)==sets[z].end())
                {
                    sets[z].push_back(y); Q.push(y);
                }
            }
        }
    }
}

int check()
{
    for(int i=0;i<z;i++)
    {
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\NITW_CD_Lab\CompilerDesignPrograms\Set_B_Programs\B8.exe
- Output Size: 2.49679660797119 MiB
- Compilation Time: 1.31s
```

```
 Note : Productions will be entered as follows : S->AB|BC represents two product
ions
and should be given one by one as S->AB and S->BC, where S will be the LHS part

and AB will be the RHS part and also A->a will have 'a' as its RHS

Enter the number of productions :
6
enter LHS of production : E
enter RHS of production : E+T
enter LHS of production : E
enter RHS of production : T
enter LHS of production : T
enter RHS of production : T*F
enter LHS of production : T
enter RHS of production : F
enter LHS of production : F
enter RHS of production : (E)
enter LHS of production : F
enter RHS of production : a
+ * ( ) a E T F
first(E) = { ( a }
first(T) = { ( a }
first(F) = { ( a }

follow(E) = { $ + ) }
follow(T) = { $ + ) * }
follow(F) = { $ + ) * }
X -> E
E -> T
T -> F
F -> a
E -> E+T
T -> T*F
F -> (E)


Sets are :

I0 { X -> .E  E -> .E+T  E -> .T  T -> .T*F  T -> .F  F -> .(E)  F -> .a  }
I1 { X -> E.  E -> E.+T  }
I2 { E -> T.  T -> T.*F  }
I3 { T -> F.  }
I4 { F -> (.E)  E -> .E+T  E -> .T  T -> .T*F  T -> .F  F -> .(E)  F -> .a  }
I5 { F -> a.  }
I6 { E -> E+.T  T -> .T*F  T -> .F  F -> .(E)  F -> .a  }
I7 { T -> T*.F  F -> .(E)  F -> .a  }
I8 { F -> (E.)  E -> E.+T  }
I9 { E -> E+T.  T -> T.*F  }
I10 { T -> T*F.  }
I11 { F -> (E).  }


Table :

   | +   *   (   )   a   $   E   T   F
----------------------------------------
0  |         4       5       1   2   3
1  | 6                   12
2  | -2  7       -2      -2
3  | -4  -4      -4      -4
4  |         4       5       8   2   3
5  | -6  -6      -6      -6
6  |         4       5           9   3
7  |         4       5               10
8  | 6           11
9  | -1  7       -1      -1
10 | -3  -3      -3      -3
11 | -5  -5      -5      -5

Enter the string :
a+a*a

$0               a+a*a$
$0a5              +a*a$
$0F3              +a*a$
$0T2              +a*a$
$0E1              +a*a$
$0E1+6             a*a$
$0E1+6a5            *a$
$0E1+6F3            *a$
$0E1+6T9            *a$
$0E1+6T9*7           a$

Accepted

--------------------------------
Process exited after 102.3 seconds with return value 0
Press any key to continue . . .
```