NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

Name – Rajat Goyal

Roll No. – 197267

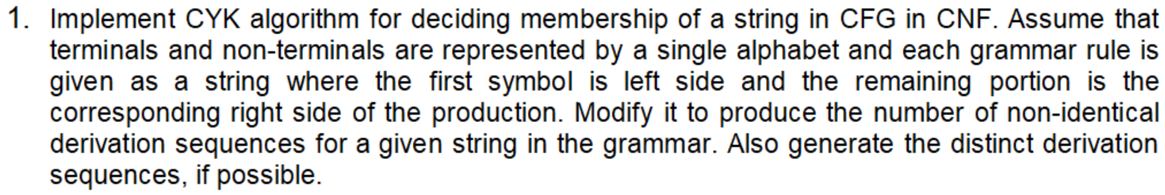
Section – B

Branch – CSE(3rd year)

LP LAB Assignment - 3

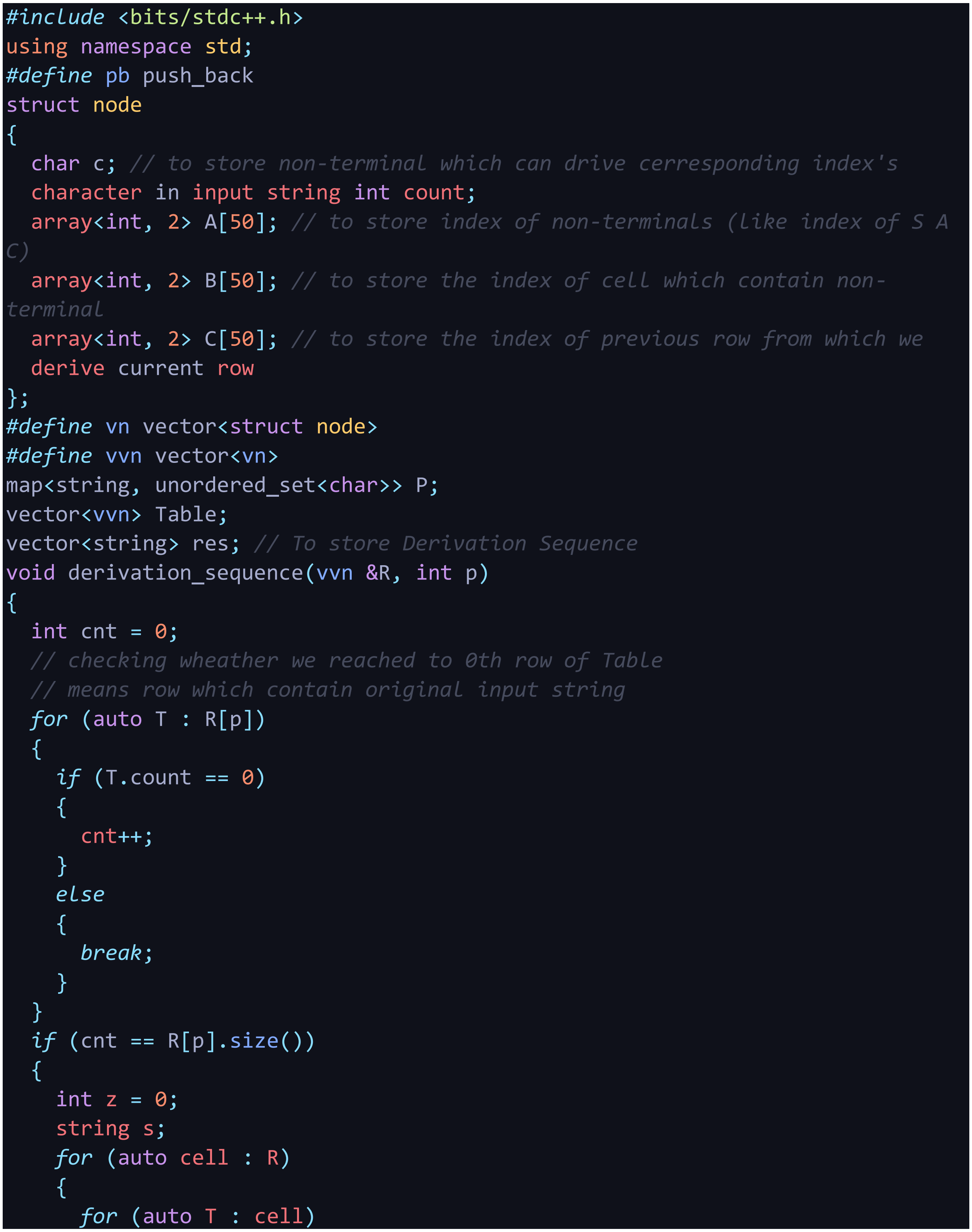
Implementing CYK Algorithm

Question:-

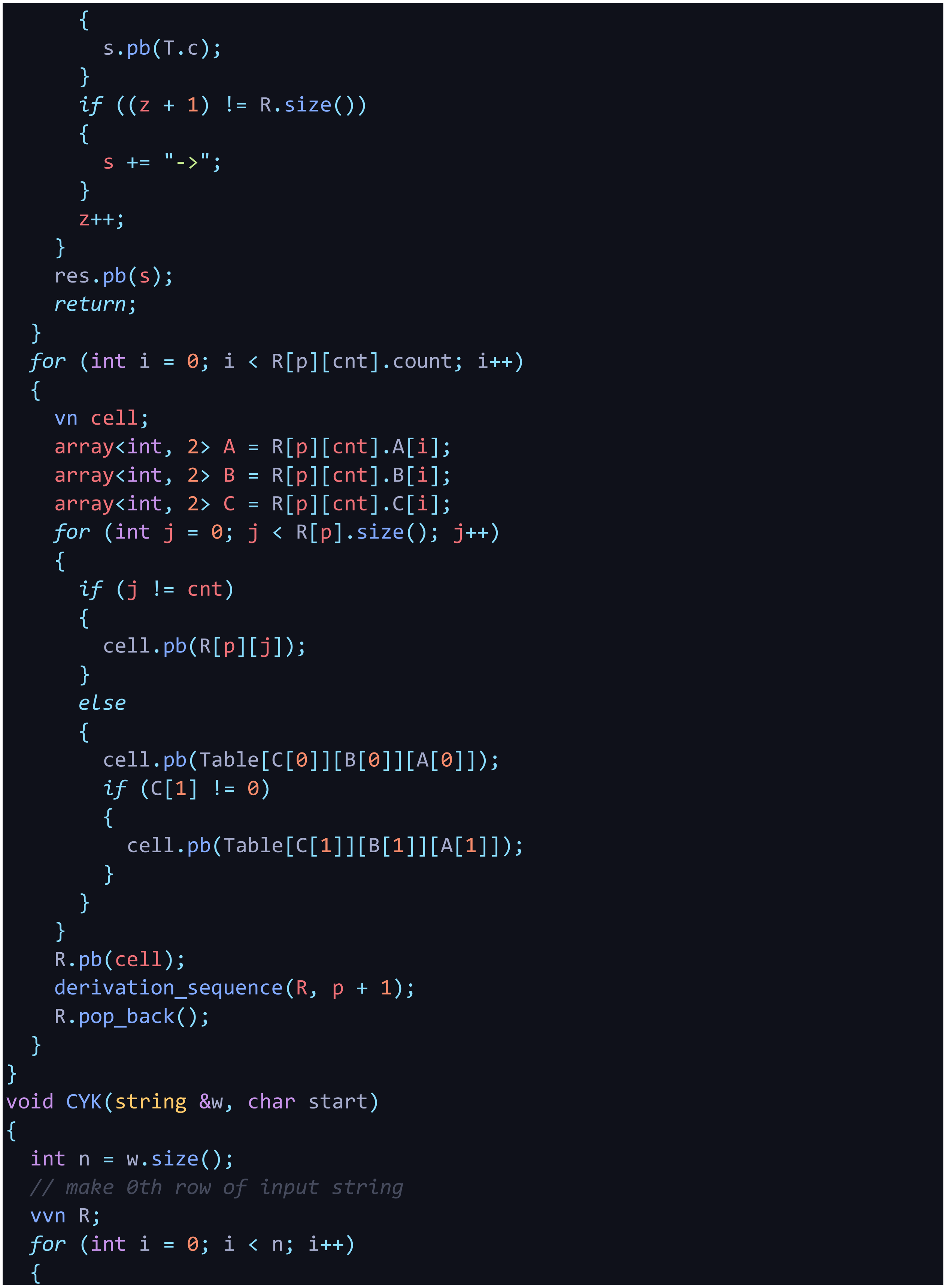


1. Implement CYK algorithm for deciding membership of a string in CFG in CNF. Assume that terminals and non-terminals are represented by a single alphabet and each grammar rule is given as a string where the first symbol is left side and the remaining portion is the corresponding right side of the production. Modify it to produce the number of non-identical derivation sequences for a given string in the grammar. Also generate the distinct derivation sequences, if possible.

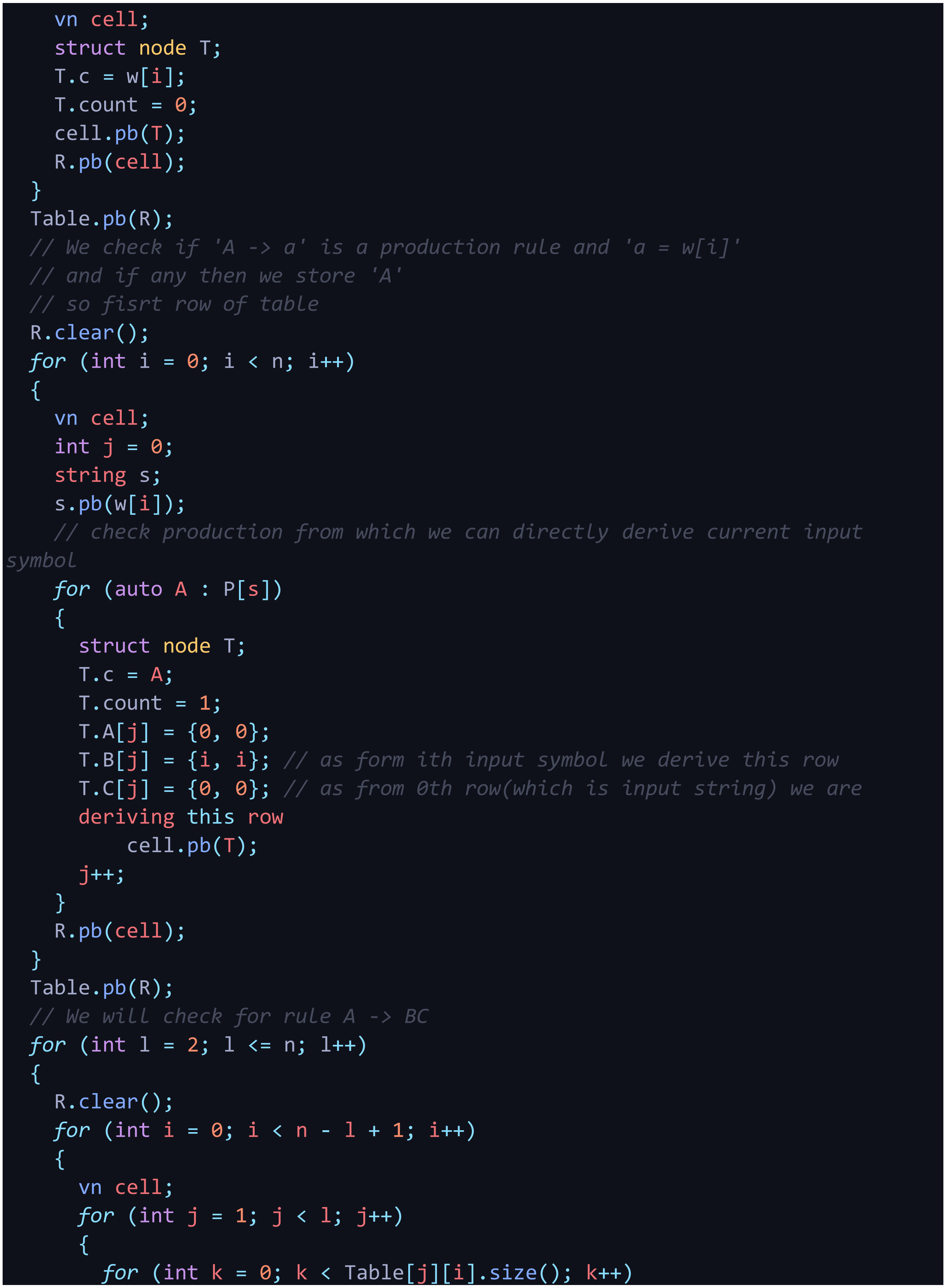
Code:-



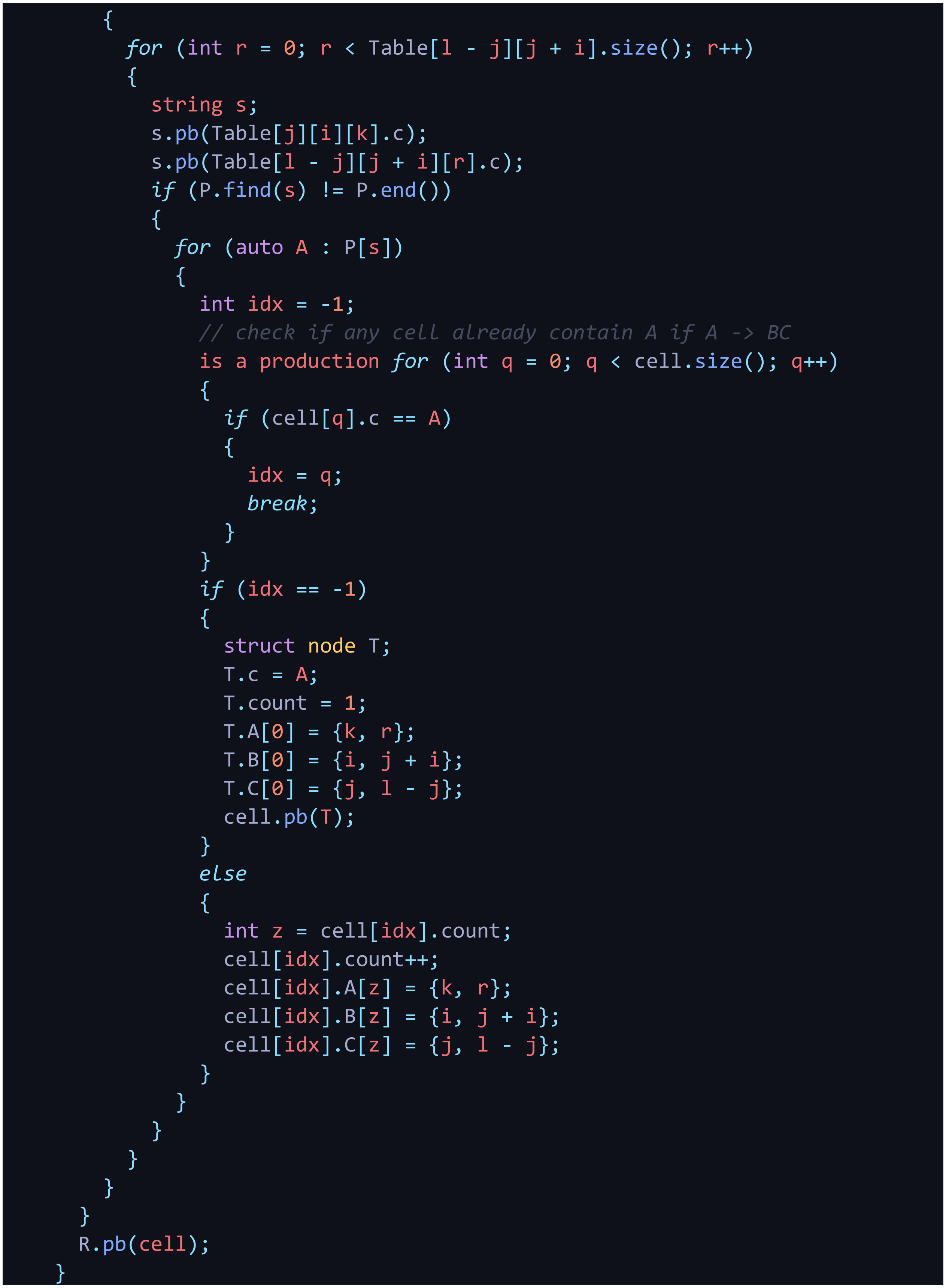
```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
struct node
{
  char c; // to store non-terminal which can drive cerresponding index's
  character in input string int count;
  array<int, 2> A[50]; // to store index of non-terminals (like index of S A
C)
  array<int, 2> B[50]; // to store the index of cell which contain non-
terminal
  array<int, 2> C[50]; // to store the index of previous row from which we
  derive current row
};
#define vn vector<struct node>
#define vvn vector<vn>
map<string, unordered_set<char>> P;
vector<vvn> Table;
vector<string> res; // To store Derivation Sequence
void derivation_sequence(vvn &R, int p)
{
  int cnt = 0;
  // checking wheather we reached to 0th row of Table
  // means row which contain original input string
  for (auto T : R[p])
  {
    if (T.count == 0)
    {
      cnt++;
    }
    else
    {
      break;
    }
  }
  if (cnt == R[p].size())
  {
    int z = 0;
    string s;
    for (auto cell : R)
    {
      for (auto T : cell)
```



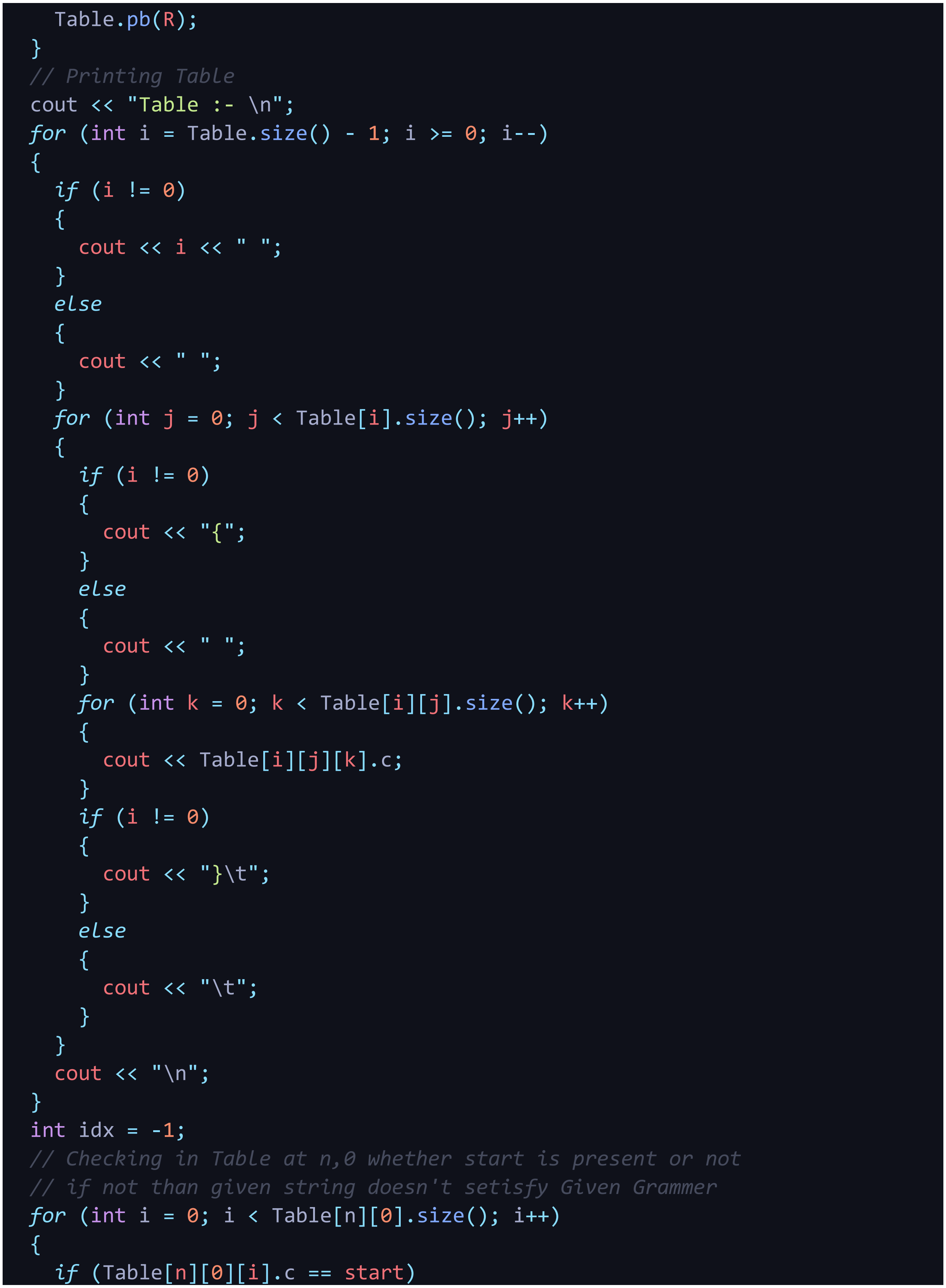
```
      {
        s.pb(T.c);
      }
      if ((z + 1) != R.size())
      {
        s += "->";
      }
      z++;
    }
    res.pb(s);
    return;
  }
  for (int i = 0; i < R[p][cnt].count; i++)
  {
    vn cell;
    array<int, 2> A = R[p][cnt].A[i];
    array<int, 2> B = R[p][cnt].B[i];
    array<int, 2> C = R[p][cnt].C[i];
    for (int j = 0; j < R[p].size(); j++)
    {
      if (j != cnt)
      {
        cell.pb(R[p][j]);
      }
      else
      {
        cell.pb(Table[C[0]][B[0]][A[0]]);
        if (C[1] != 0)
        {
          cell.pb(Table[C[1]][B[1]][A[1]]);
        }
      }
    }
    R.pb(cell);
    derivation_sequence(R, p + 1);
    R.pop_back();
  }
}
void CYK(string &w, char start)
{
  int n = w.size();
  // make 0th row of input string
  vvn R;
  for (int i = 0; i < n; i++)
  {
```



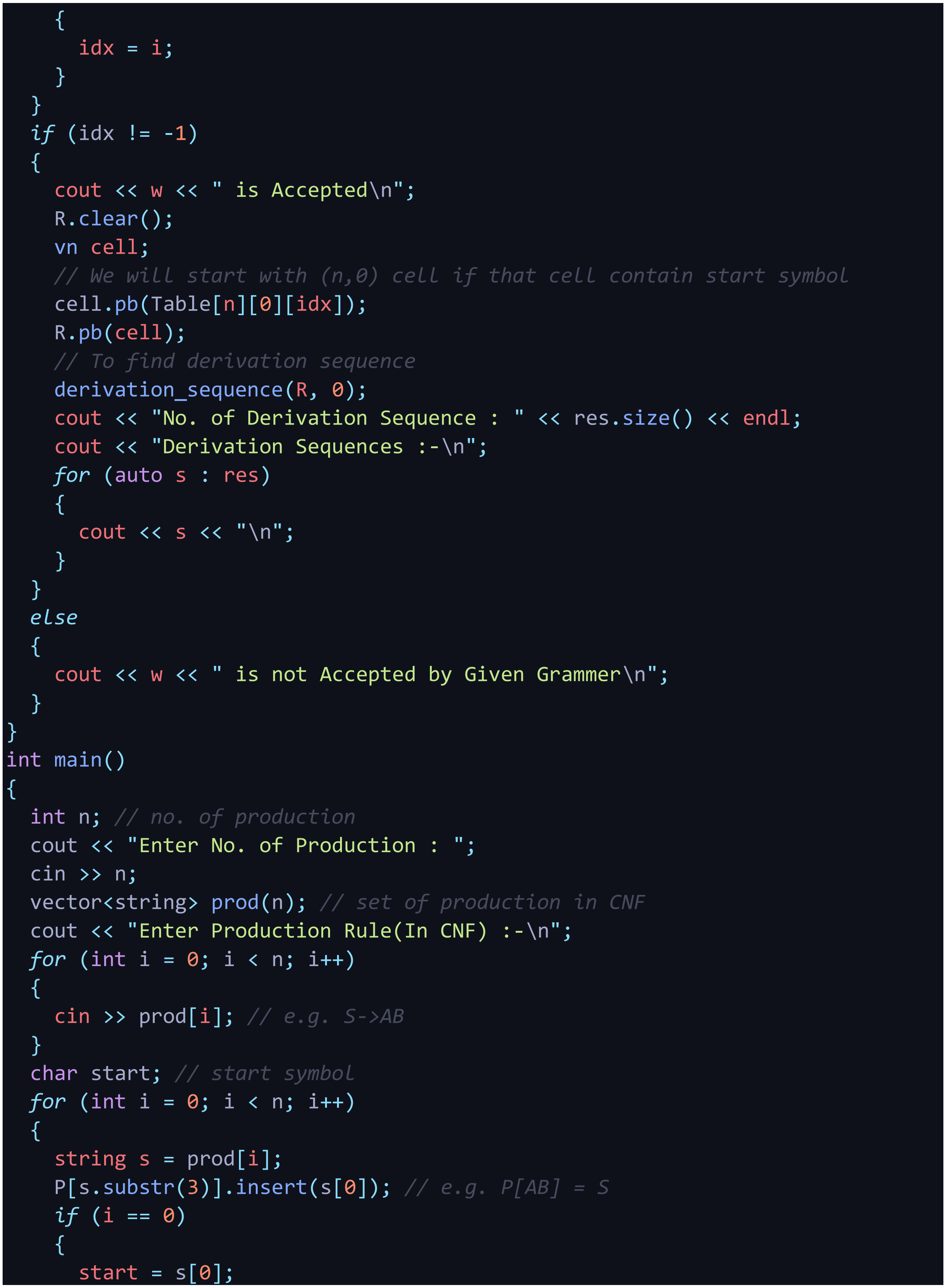
```
    vn cell;
    struct node T;
    T.c = w[i];
    T.count = 0;
    cell.pb(T);
    R.pb(cell);
  }
  Table.pb(R);
  // We check if 'A -> a' is a production rule and 'a = w[i]'
  // and if any then we store 'A'
  // so fisrt row of table
  R.clear();
  for (int i = 0; i < n; i++)
  {
    vn cell;
    int j = 0;
    string s;
    s.pb(w[i]);
    // check production from which we can directly derive current input 
symbol
    for (auto A : P[s])
    {
      struct node T;
      T.c = A;
      T.count = 1;
      T.A[j] = {0, 0};
      T.B[j] = {i, i}; // as form ith input symbol we derive this row
      T.C[j] = {0, 0}; // as from 0th row(which is input string) we are 
      deriving this row
          cell.pb(T);
      j++;
    }
    R.pb(cell);
  }
  Table.pb(R);
  // We will check for rule A -> BC
  for (int l = 2; l <= n; l++)
  {
    R.clear();
    for (int i = 0; i < n - l + 1; i++)
    {
      vn cell;
      for (int j = 1; j < l; j++)
      {
        for (int k = 0; k < Table[j][i].size(); k++)
```



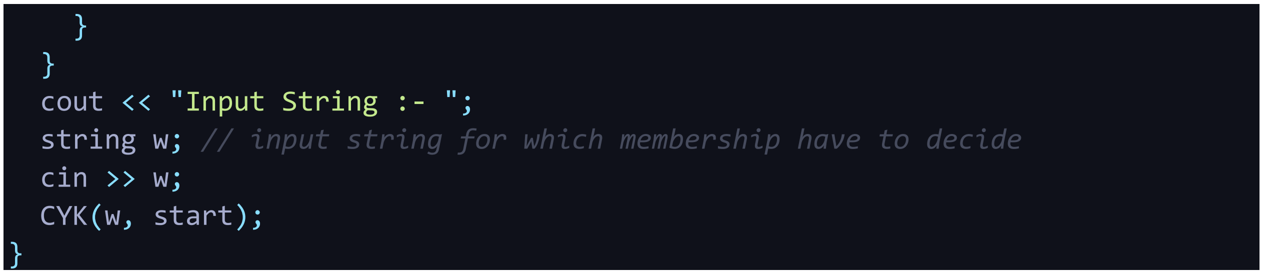
```
        {
          for (int r = 0; r < Table[l - j][j + i].size(); r++)
          {
            string s;
            s.pb(Table[j][i][k].c);
            s.pb(Table[l - j][j + i][r].c);
            if (P.find(s) != P.end())
            {
              for (auto A : P[s])
              {
                int idx = -1;
                // check if any cell already contain A if A -> BC
                is a production for (int q = 0; q < cell.size(); q++)
                {
                  if (cell[q].c == A)
                  {
                    idx = q;
                    break;
                  }
                }
                if (idx == -1)
                {
                  struct node T;
                  T.c = A;
                  T.count = 1;
                  T.A[0] = {k, r};
                  T.B[0] = {i, j + i};
                  T.C[0] = {j, l - j};
                  cell.pb(T);
                }
                else
                {
                  int z = cell[idx].count;
                  cell[idx].count++;
                  cell[idx].A[z] = {k, r};
                  cell[idx].B[z] = {i, j + i};
                  cell[idx].C[z] = {j, l - j};
                }
              }
            }
          }
        }
      }
      R.pb(cell);
    }
```



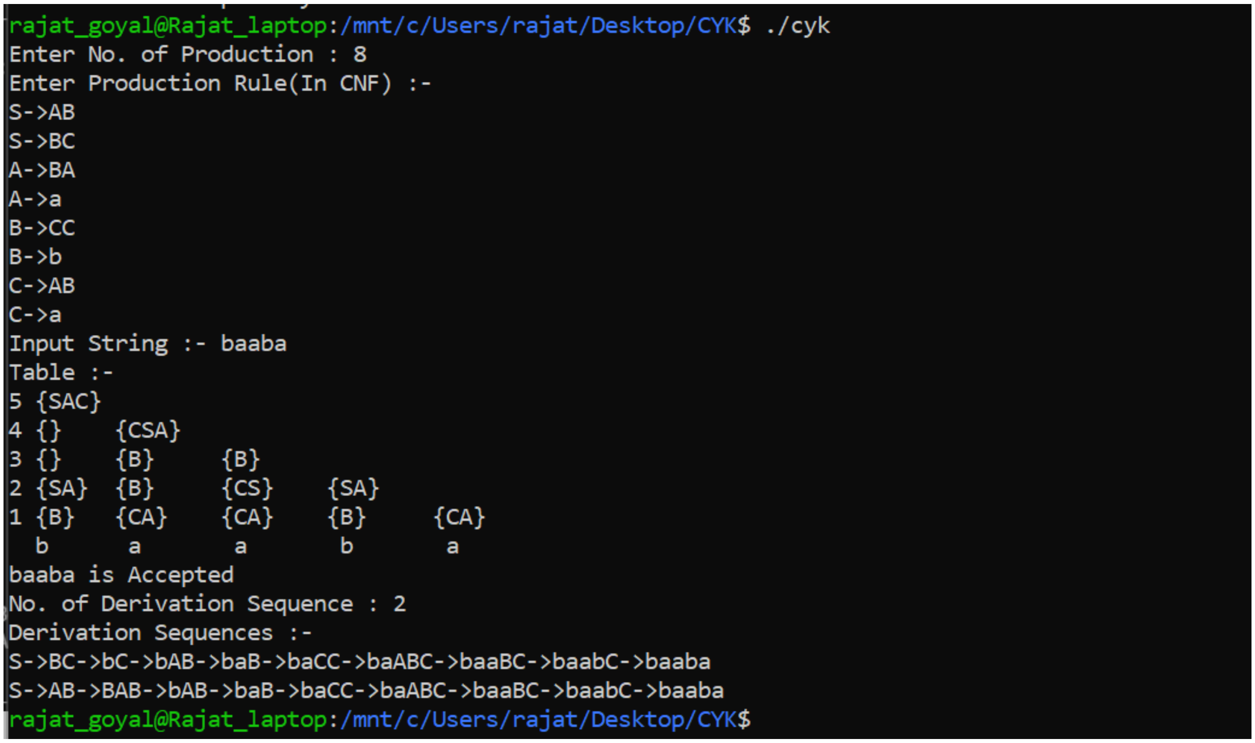
```
  Table.pb(R);
}
// Printing Table
cout << "Table :- \n";
for (int i = Table.size() - 1; i >= 0; i--)
{
  if (i != 0)
  {
    cout << i << " ";
  }
  else
  {
    cout << " ";
  }
  for (int j = 0; j < Table[i].size(); j++)
  {
    if (i != 0)
    {
      cout << "{";
    }
    else
    {
      cout << " ";
    }
    for (int k = 0; k < Table[i][j].size(); k++)
    {
      cout << Table[i][j][k].c;
    }
    if (i != 0)
    {
      cout << "}\t";
    }
    else
    {
      cout << "\t";
    }
  }
  cout << "\n";
}
int idx = -1;
// Checking in Table at n,0 whether start is present or not
// if not than given string doesn't setisfy Given Grammer
for (int i = 0; i < Table[n][0].size(); i++)
{
  if (Table[n][0][i].c == start)
```



```
    {
      idx = i;
    }
  }
  if (idx != -1)
  {
    cout << w << " is Accepted\n";
    R.clear();
    vn cell;
    // We will start with (n,0) cell if that cell contain start symbol
    cell.pb(Table[n][0][idx]);
    R.pb(cell);
    // To find derivation sequence
    derivation_sequence(R, 0);
    cout << "No. of Derivation Sequence : " << res.size() << endl;
    cout << "Derivation Sequences :-\n";
    for (auto s : res)
    {
      cout << s << "\n";
    }
  }
  else
  {
    cout << w << " is not Accepted by Given Grammer\n";
  }
}
int main()
{
  int n; // no. of production
  cout << "Enter No. of Production : ";
  cin >> n;
  vector<string> prod(n); // set of production in CNF
  cout << "Enter Production Rule(In CNF) :-\n";
  for (int i = 0; i < n; i++)
  {
    cin >> prod[i]; // e.g. S->AB
  }
  char start; // start symbol
  for (int i = 0; i < n; i++)
  {
    string s = prod[i];
    P[s.substr(3)].insert(s[0]); // e.g. P[AB] = S
    if (i == 0)
    {
      start = s[0];
```



```
    }
  }
  cout << "Input String :- ";
  string w; // input string for which membership have to decide
  cin >> w;
  CYK(w, start);
}
```



```
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/CYK$ ./cyk
Enter No. of Production : 8
Enter Production Rule(In CNF) :-
S->AB
S->BC
A->BA
A->a
B->CC
B->b
C->AB
C->a
Input String :- baaba
Table :-
5 {SAC}
4 {}    {CSA}
3 {}    {B}     {B}
2 {SA}  {B}     {CS}    {SA}
1 {B}   {CA}    {CA}    {B}     {CA}
   b      a       a       b       a
baaba is Accepted
No. of Derivation Sequence : 2
Derivation Sequences :-
S->BC->bC->bAB->baB->baCC->baABC->baaBC->baabC->baaba
S->AB->BAB->bAB->baB->baCC->baABC->baaBC->baabC->baaba
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/CYK$
```