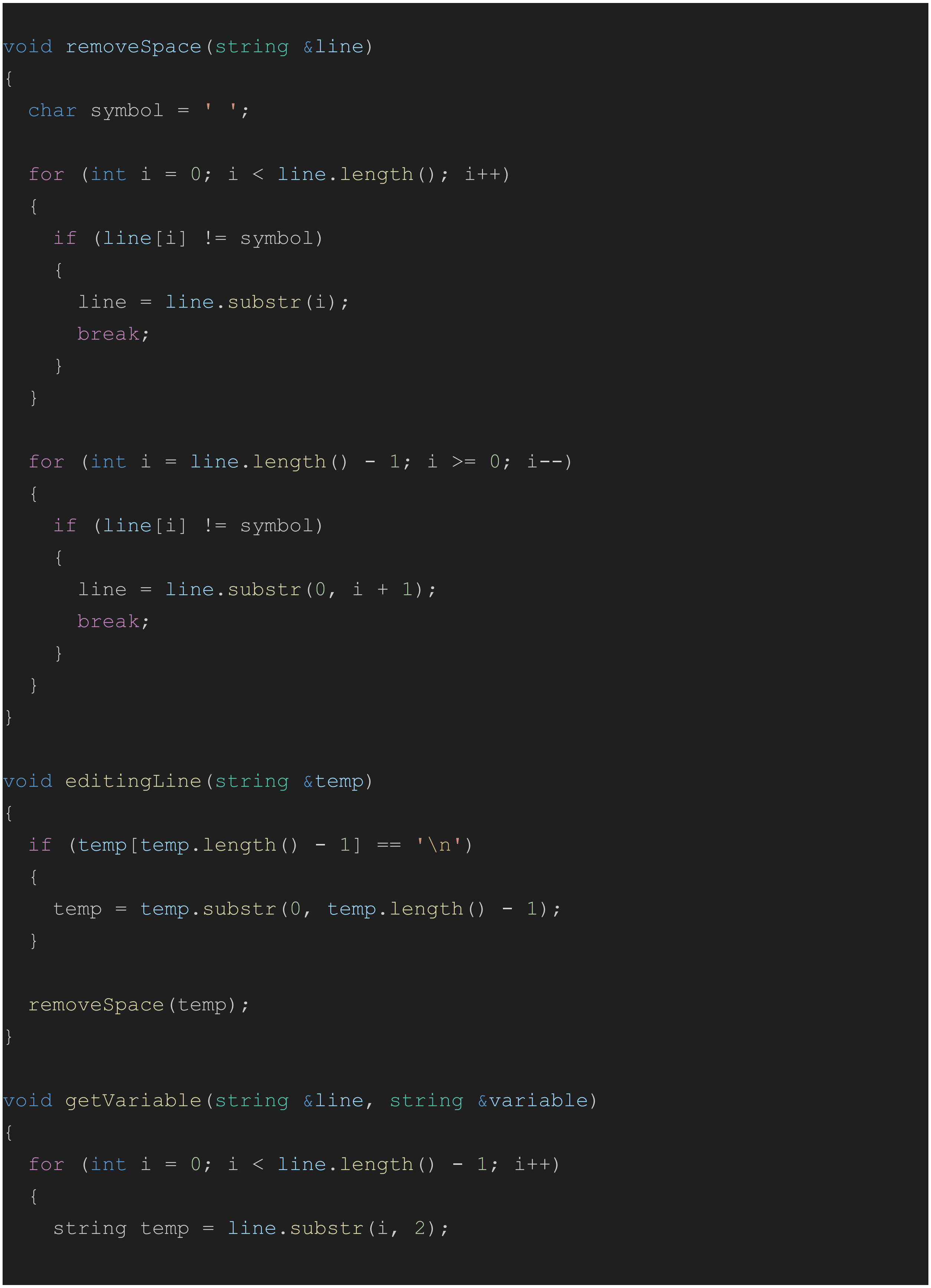
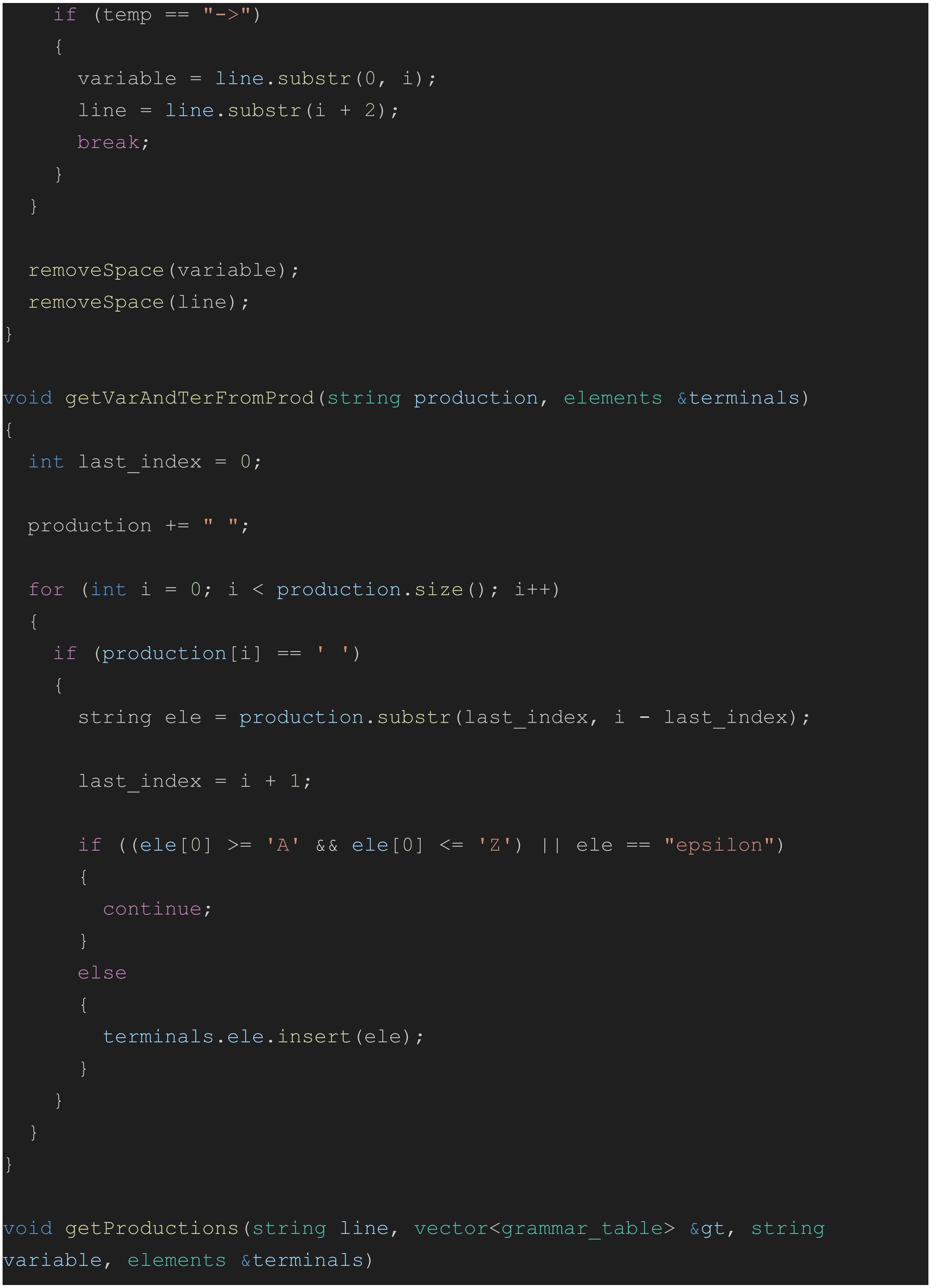
LAB 8 SLR PARSER
COMPILER DESIGN LAB
22BCE5155
SAI DHAKSHAN Y

Code :

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

typedef struct
{
  string variable;
  string production;
} grammar_table;

typedef struct
{
  string row_name;
  vector<string> columns;
} table;

typedef struct
{
  unordered_set<string> ele;
  int count;
} elements;

typedef struct
{
  vector<string> var;
  vector<string> prod;
  vector<int> next;
  int originalNo;
} state;
```

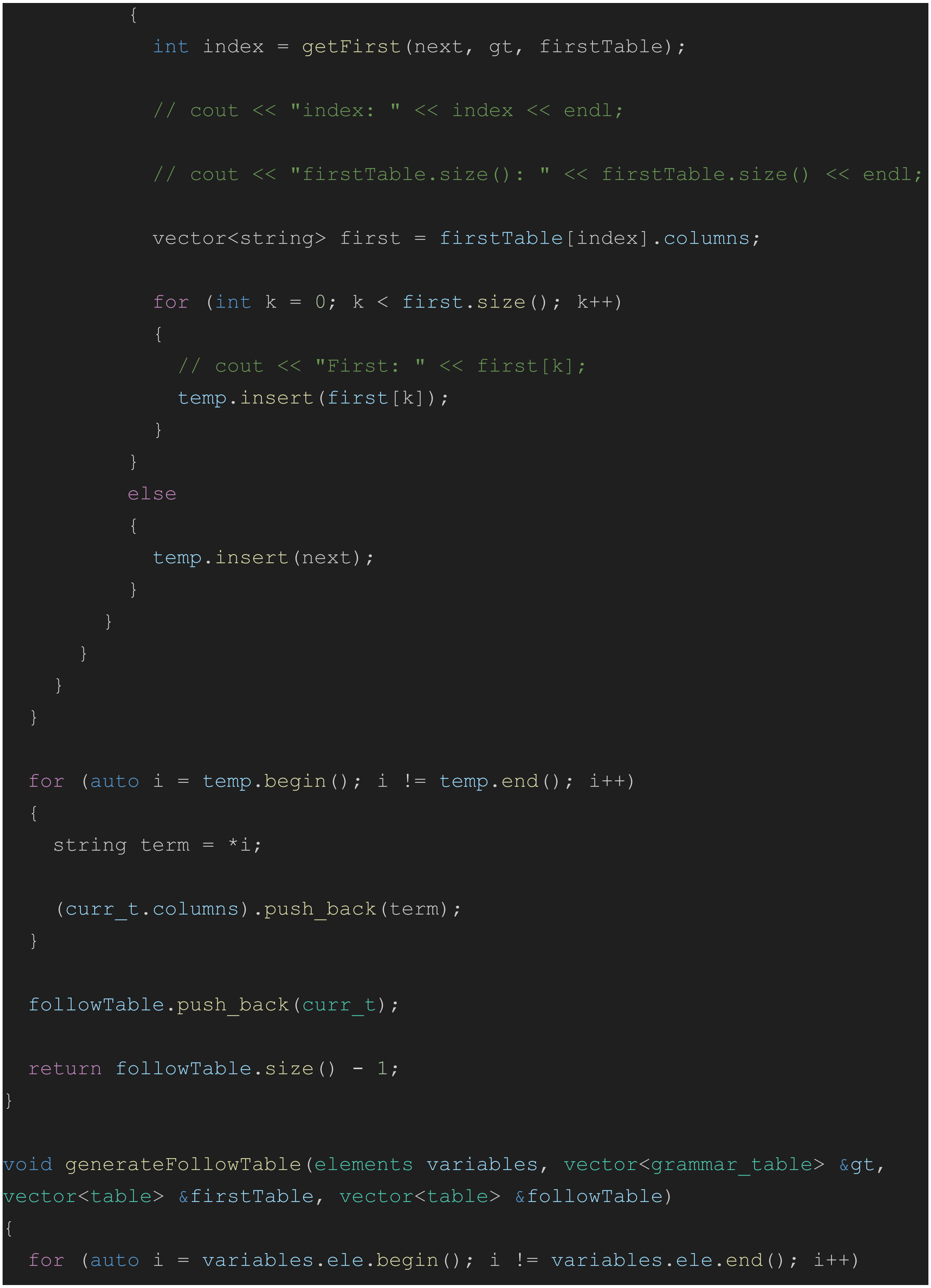


```
void removeSpace(string &line)
{
  char symbol = ' ';

  for (int i = 0; i < line.length(); i++)
  {
    if (line[i] != symbol)
    {
      line = line.substr(i);
      break;
    }
  }

  for (int i = line.length() - 1; i >= 0; i--)
  {
    if (line[i] != symbol)
    {
      line = line.substr(0, i + 1);
      break;
    }
  }
}

void editingLine(string &temp)
{
  if (temp[temp.length() - 1] == '\n')
  {
    temp = temp.substr(0, temp.length() - 1);
  }

  removeSpace(temp);
}

void getVariable(string &line, string &variable)
{
  for (int i = 0; i < line.length() - 1; i++)
  {
    string temp = line.substr(i, 2);
```

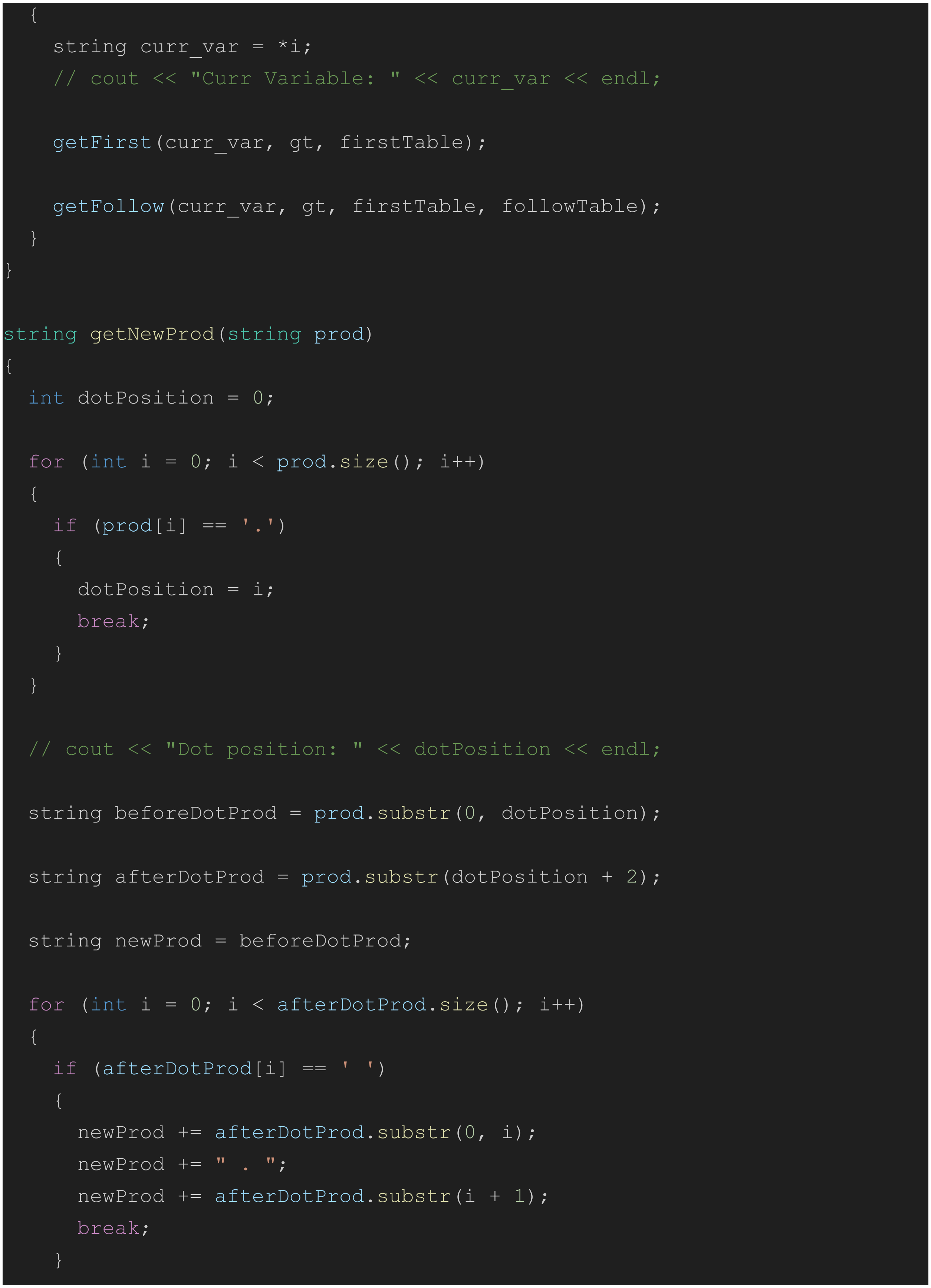


```
    if (temp == "->")
    {
      variable = line.substr(0, i);
      line = line.substr(i + 2);
      break;
    }
  }

  removeSpace(variable);
  removeSpace(line);
}

void getVarAndTerFromProd(string production, elements &terminals)
{
  int last_index = 0;

  production += " ";

  for (int i = 0; i < production.size(); i++)
  {
    if (production[i] == ' ')
    {
      string ele = production.substr(last_index, i - last_index);

      last_index = i + 1;

      if ((ele[0] >= 'A' && ele[0] <= 'Z') || ele == "epsilon")
      {
        continue;
      }
      else
      {
        terminals.ele.insert(ele);
      }
    }
  }
}

void getProductions(string line, vector<grammar_table> &gt, string
variable, elements &terminals)
```

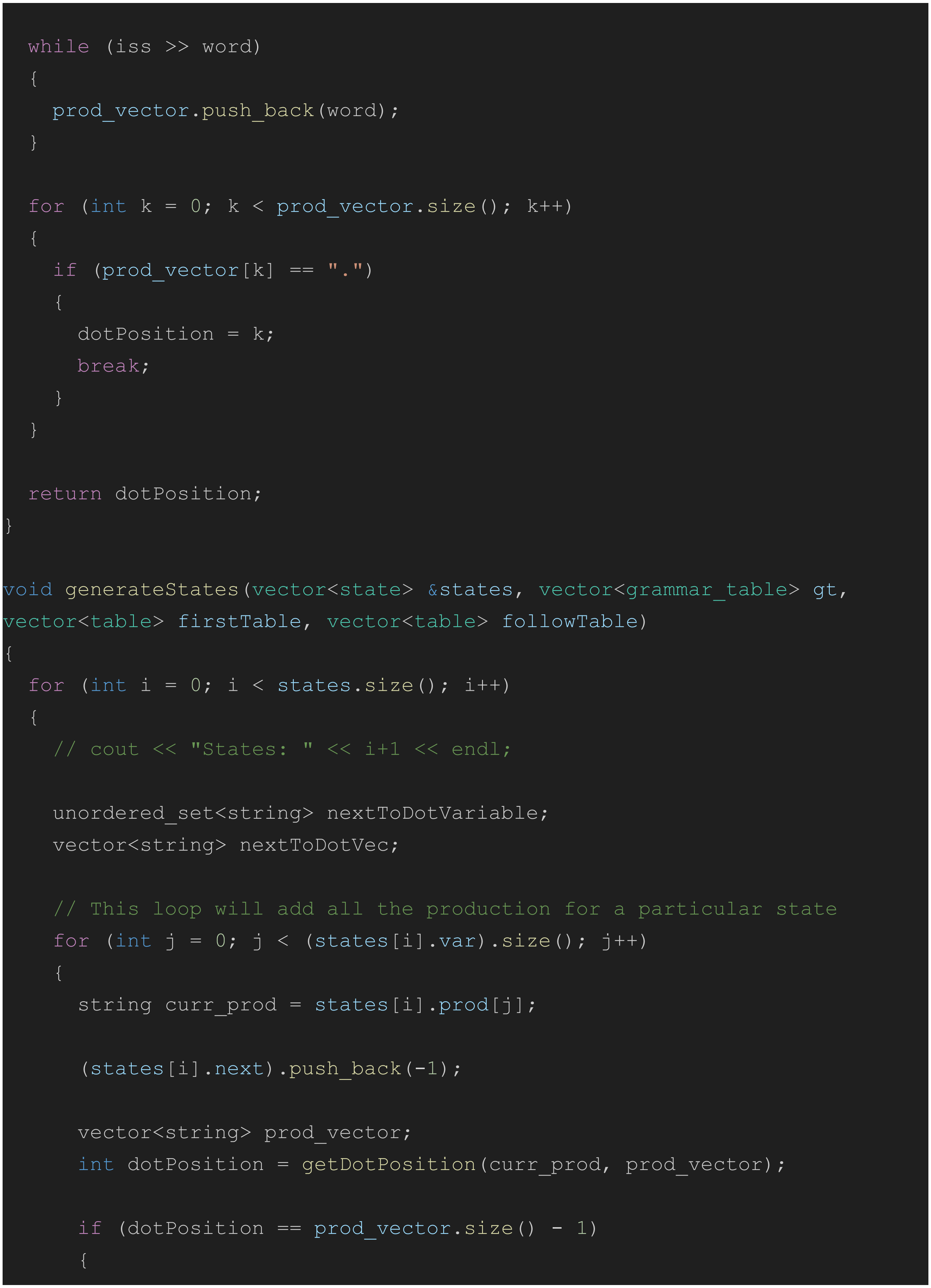
```
{
  string production = "";
  int last_index = 0;

  for (int i = 0; i < line.length(); i++)
  {
    char temp = line[i];

    if (temp == '|')
    {
      production = line.substr(last_index, i - last_index);
      last_index = i + 1;

      removeSpace(production);

      // write the function to process terminals in the production
      getVarAndTerFromProd(production, terminals);

      grammar_table curr_g;

      curr_g.variable = variable;

      curr_g.production = production;

      gt.push_back(curr_g);
    }
  }
}

void processLine(string line, vector<grammar_table> &gt, elements &vars,
elements &terminals)
{
  // grammar_table curr_g;

  string variable;
  getVariable(line, variable);

  vars.ele.insert(variable);

  vars.count += 1;
  line += " |";

  getProductions(line, gt, variable, terminals);
}

int getFirst(string next, vector<grammar_table> &gt, vector<table>
&firstTable)
{
  for (int i = 0; i < firstTable.size(); i++)
  {
    string var = firstTable[i].row_name;

    if (var == next)
    {
      return i;
    }
  }

  table curr_t;

  curr_t.row_name = next;
  unordered_set<string> temp2;

  for (int i = 0; i < gt.size(); i++)
  {
    if (gt[i].variable != next)
    {
      continue;
    }

    string prod = gt[i].production;
    string temp = "";
    temp += prod[0];

    // cout << "First, Curr production: " << prod << endl;

    if (prod[0] >= 'A' && prod[0] <= 'Z')
    {
      if (temp == next)
      {
        continue;
      }
      int index = getFirst(temp, gt, firstTable);

      vector<string> first = firstTable[index].columns;

      for (int k = 0; k < first.size(); k++)
      {
        temp2.insert(first[k]);
      }
    }
    else
    {
      temp2.insert(temp);
    }
  }

  for (auto i = temp2.begin(); i != temp2.end(); i++)
  {
    string term = *i;

    (curr_t.columns).push_back(term);
  }

  firstTable.push_back(curr_t);

  return firstTable.size() - 1;
}

int getFollow(string variable, vector<grammar_table> &gt, vector<table>
&firstTable, vector<table> &followTable)
{
  for (int i = 0; i < followTable.size(); i++)
  {
    string var = followTable[i].row_name;

    if (var == variable)
    {
      return i;
    }
  }

  table curr_t;

  curr_t.row_name = variable;
  unordered_set<string> temp;

  for (int i = 0; i < gt.size(); i++)
  {
    string prod = gt[i].production;

    // cout << "Follow, Curr production: " << prod << endl;
    for (int j = 0; j < prod.length(); j++)
    {
      if (prod[j] == variable[0])
      {
        if (j == prod.length() - 1)
        {
          if (prod[j] == gt[i].variable[0])
          {
            continue;
          }

          int index = getFollow(gt[i].variable, gt, firstTable,
followTable);

          vector<string> follow = followTable[index].columns;

          for (int k = 0; k < follow.size(); k++)
          {
            temp.insert(follow[k]);
          }
        }
        else
        {
          string next = "";
          next += prod[j + 2];

          if (next[0] >= 'A' && next[0] <= 'Z')
```



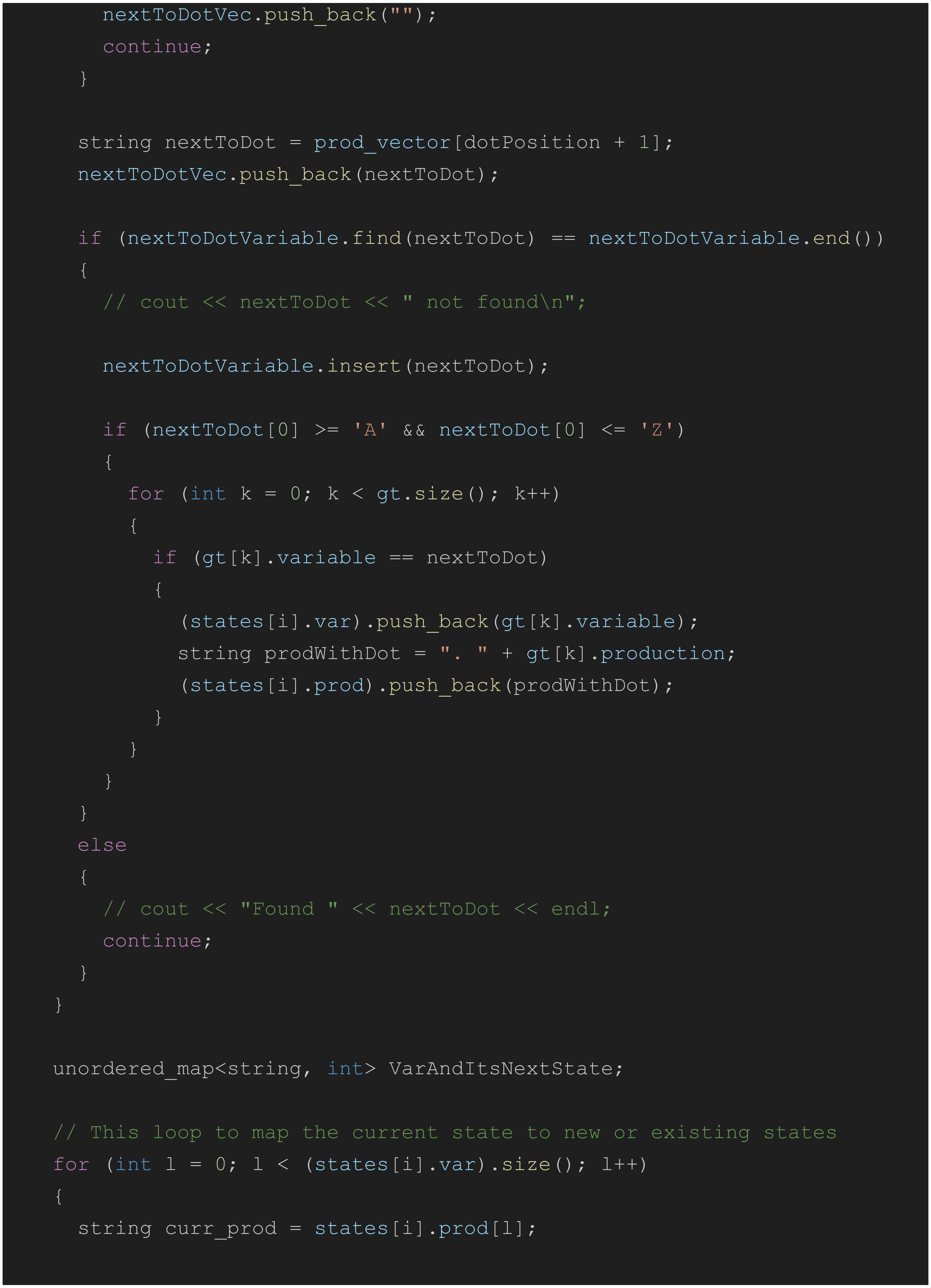
```
          {
            int index = getFirst(next, gt, firstTable);

            // cout << "index: " << index << endl;

            // cout << "firstTable.size(): " << firstTable.size() << endl;

            vector<string> first = firstTable[index].columns;

            for (int k = 0; k < first.size(); k++)
            {
              // cout << "First: " << first[k];
              temp.insert(first[k]);
            }
          }
          else
          {
            temp.insert(next);
          }
        }
      }
    }
  }

  for (auto i = temp.begin(); i != temp.end(); i++)
  {
    string term = *i;

    (curr_t.columns).push_back(term);
  }

  followTable.push_back(curr_t);

  return followTable.size() - 1;
}

void generateFollowTable(elements variables, vector<grammar_table> &gt,
vector<table> &firstTable, vector<table> &followTable)
{
  for (auto i = variables.ele.begin(); i != variables.ele.end(); i++)
```



```
  {
    string curr_var = *i;
    // cout << "Curr Variable: " << curr_var << endl;

    getFirst(curr_var, gt, firstTable);

    getFollow(curr_var, gt, firstTable, followTable);
  }
}

string getNewProd(string prod)
{
  int dotPosition = 0;

  for (int i = 0; i < prod.size(); i++)
  {
    if (prod[i] == '.')
    {
      dotPosition = i;
      break;
    }
  }

  // cout << "Dot position: " << dotPosition << endl;

  string beforeDotProd = prod.substr(0, dotPosition);

  string afterDotProd = prod.substr(dotPosition + 2);

  string newProd = beforeDotProd;

  for (int i = 0; i < afterDotProd.size(); i++)
  {
    if (afterDotProd[i] == ' ')
    {
      newProd += afterDotProd.substr(0, i);
      newProd += " . ";
      newProd += afterDotProd.substr(i + 1);
      break;
    }
    else if (i == afterDotProd.size() - 1)
    {
      newProd += afterDotProd.substr(0, i + 1);
      newProd += " .";
      break;
    }
  }

  // cout << "///" << newProd << "///" << endl;

  return newProd;
}

int findIfAlreadyExists(vector<state> states, string nextToDot, string
newProd)
{
  for (int i = 0; i < states.size(); i++)
  {
    int size = states[i].originalNo;

    for (int j = 0; j < size; j++)
    {
      if ((states[i].var[j] == nextToDot) && (states[i].prod[j] ==
newProd))
      {
        return i;
      }
    }
  }

  return -1;
}

int getDotPosition(string curr_prod, vector<string> &prod_vector)
{
  istringstream iss(curr_prod);
  string word;

  int dotPosition = 0;
```

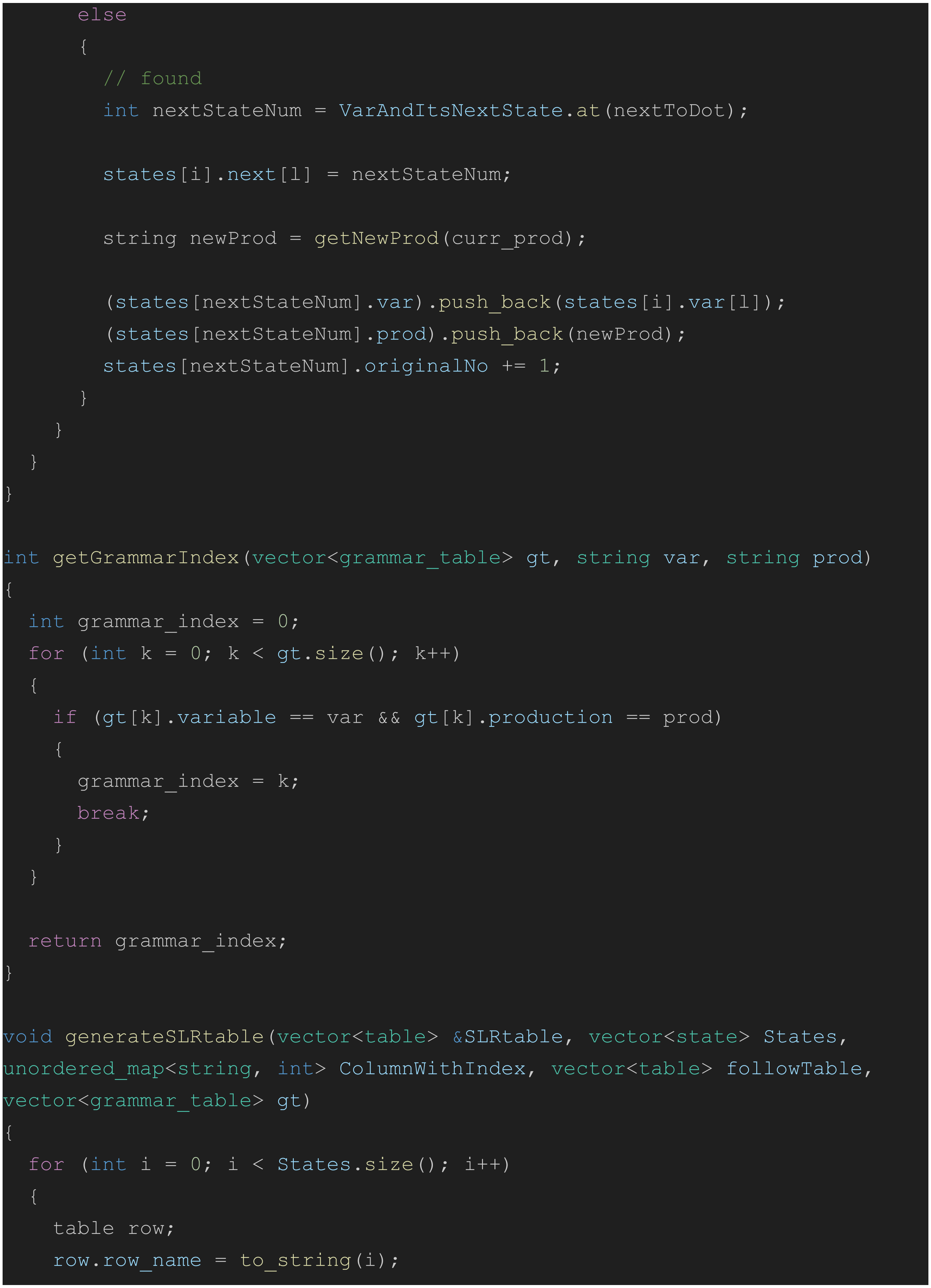


```
  while (iss >> word)
  {
    prod_vector.push_back(word);
  }

  for (int k = 0; k < prod_vector.size(); k++)
  {
    if (prod_vector[k] == ".")
    {
      dotPosition = k;
      break;
    }
  }

  return dotPosition;
}

void generateStates(vector<state> &states, vector<grammar_table> gt,
vector<table> firstTable, vector<table> followTable)
{
  for (int i = 0; i < states.size(); i++)
  {
    // cout << "States: " << i+1 << endl;

    unordered_set<string> nextToDotVariable;
    vector<string> nextToDotVec;

    // This loop will add all the production for a particular state
    for (int j = 0; j < (states[i].var).size(); j++)
    {
      string curr_prod = states[i].prod[j];

      (states[i].next).push_back(-1);

      vector<string> prod_vector;
      int dotPosition = getDotPosition(curr_prod, prod_vector);

      if (dotPosition == prod_vector.size() - 1)
      {
```

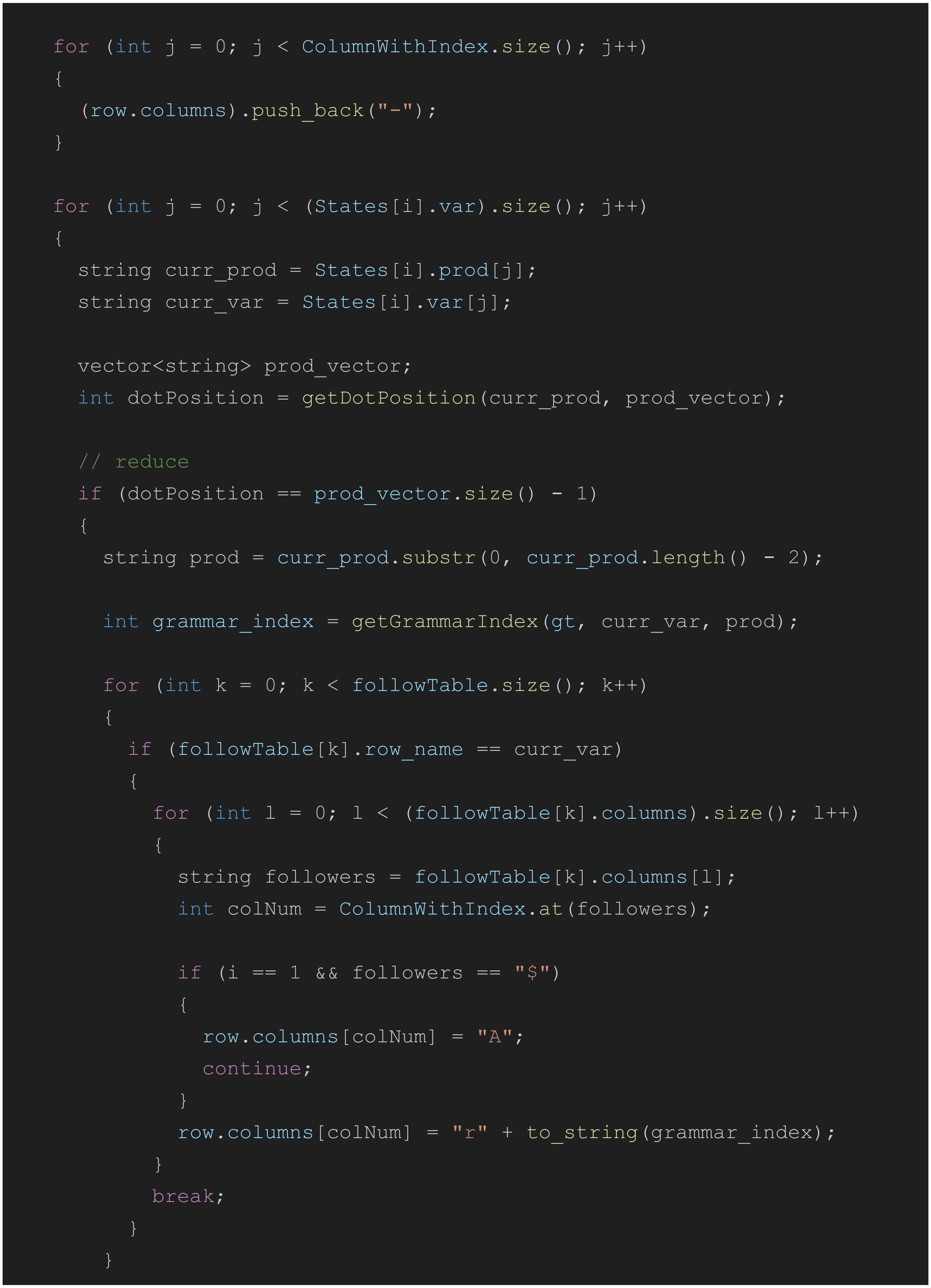


```
    nextToDotVec.push_back("");
    continue;
  }

  string nextToDot = prod_vector[dotPosition + 1];
  nextToDotVec.push_back(nextToDot);

  if (nextToDotVariable.find(nextToDot) == nextToDotVariable.end())
  {
    // cout << nextToDot << " not found\n";

    nextToDotVariable.insert(nextToDot);

    if (nextToDot[0] >= 'A' && nextToDot[0] <= 'Z')
    {
      for (int k = 0; k < gt.size(); k++)
      {
        if (gt[k].variable == nextToDot)
        {
          (states[i].var).push_back(gt[k].variable);
          string prodWithDot = ". " + gt[k].production;
          (states[i].prod).push_back(prodWithDot);
        }
      }
    }
  }
  else
  {
    // cout << "Found " << nextToDot << endl;
    continue;
  }
}

unordered_map<string, int> VarAndItsNextState;

// This loop to map the current state to new or existing states
for (int l = 0; l < (states[i].var).size(); l++)
{
  string curr_prod = states[i].prod[l];
```

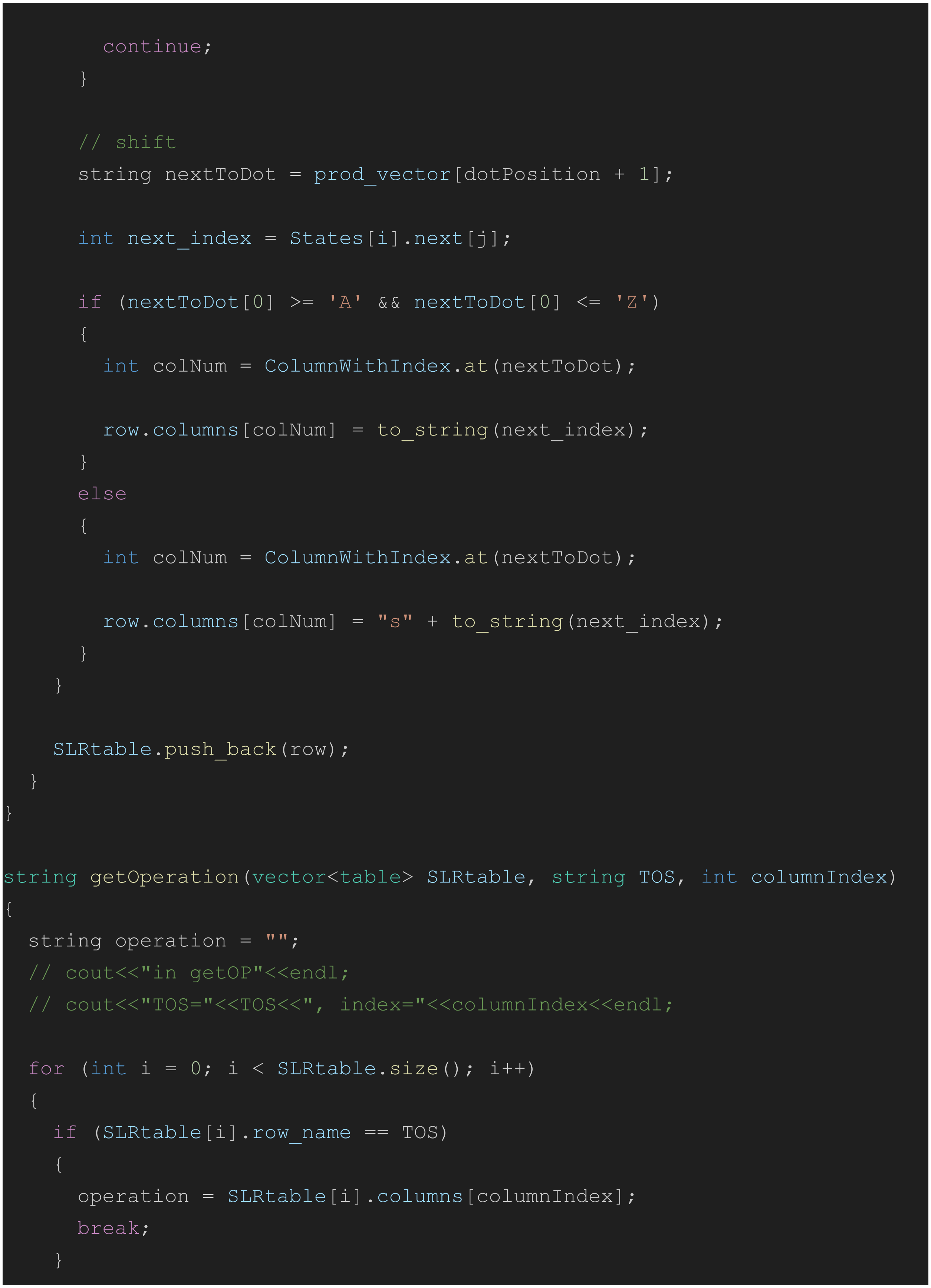
```
      string nextToDot = nextToDotVec[l];
      // cout << "curr_prod: " << curr_prod << endl;
      // cout << "nextToDot: " << nextToDot << endl;

      if (nextToDot == "")
      {
        continue;
      }

      string newProd = getNewProd(curr_prod);

      // cout << "newProd: " << newProd << endl;

      if (VarAndItsNextState.find(nextToDot) == VarAndItsNextState.end())
      {
        // Not found

        int stateNum = findIfAlreadyExists(states, states[i].var[l],
newProd);

        if (stateNum != -1)
        {
          states[i].next[l] = (stateNum);
          continue;
        }

        int newStateNum = states.size();

        VarAndItsNextState[nextToDot] = newStateNum;

        states[i].next[l] = newStateNum;

        state newState;

        (newState.var).push_back(states[i].var[l]);
        (newState.prod).push_back(newProd);
        newState.originalNo = 1;

        states.push_back(newState);
      }
```



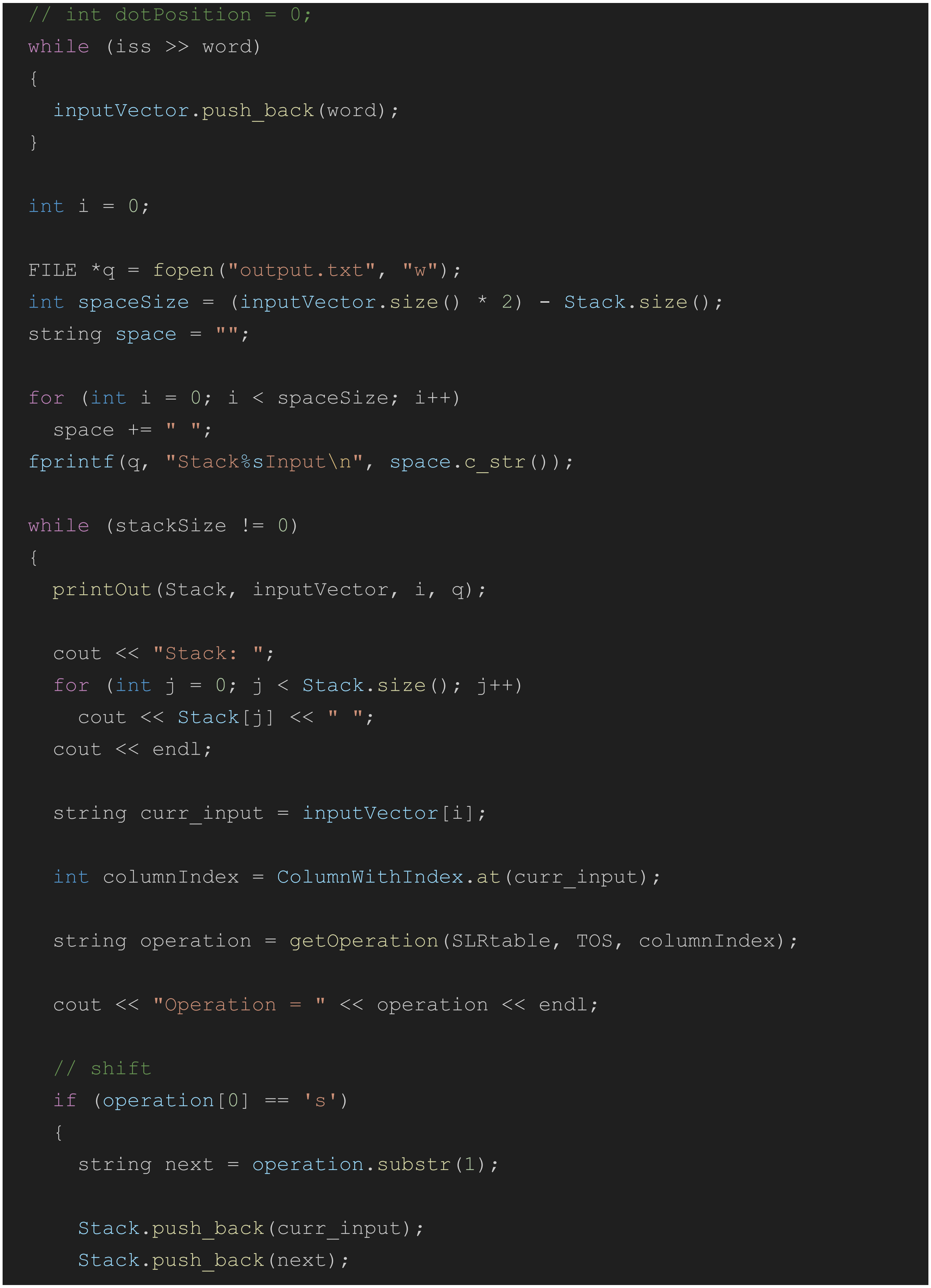
```
      else
      {
        // found
        int nextStateNum = VarAndItsNextState.at(nextToDot);

        states[i].next[l] = nextStateNum;

        string newProd = getNewProd(curr_prod);

        (states[nextStateNum].var).push_back(states[i].var[l]);
        (states[nextStateNum].prod).push_back(newProd);
        states[nextStateNum].originalNo += 1;
      }
    }
  }
}

int getGrammarIndex(vector<grammar_table> gt, string var, string prod)
{
  int grammar_index = 0;
  for (int k = 0; k < gt.size(); k++)
  {
    if (gt[k].variable == var && gt[k].production == prod)
    {
      grammar_index = k;
      break;
    }
  }

  return grammar_index;
}

void generateSLRtable(vector<table> &SLRtable, vector<state> States,
unordered_map<string, int> ColumnWithIndex, vector<table> followTable,
vector<grammar_table> gt)
{
  for (int i = 0; i < States.size(); i++)
  {
    table row;
    row.row_name = to_string(i);
```



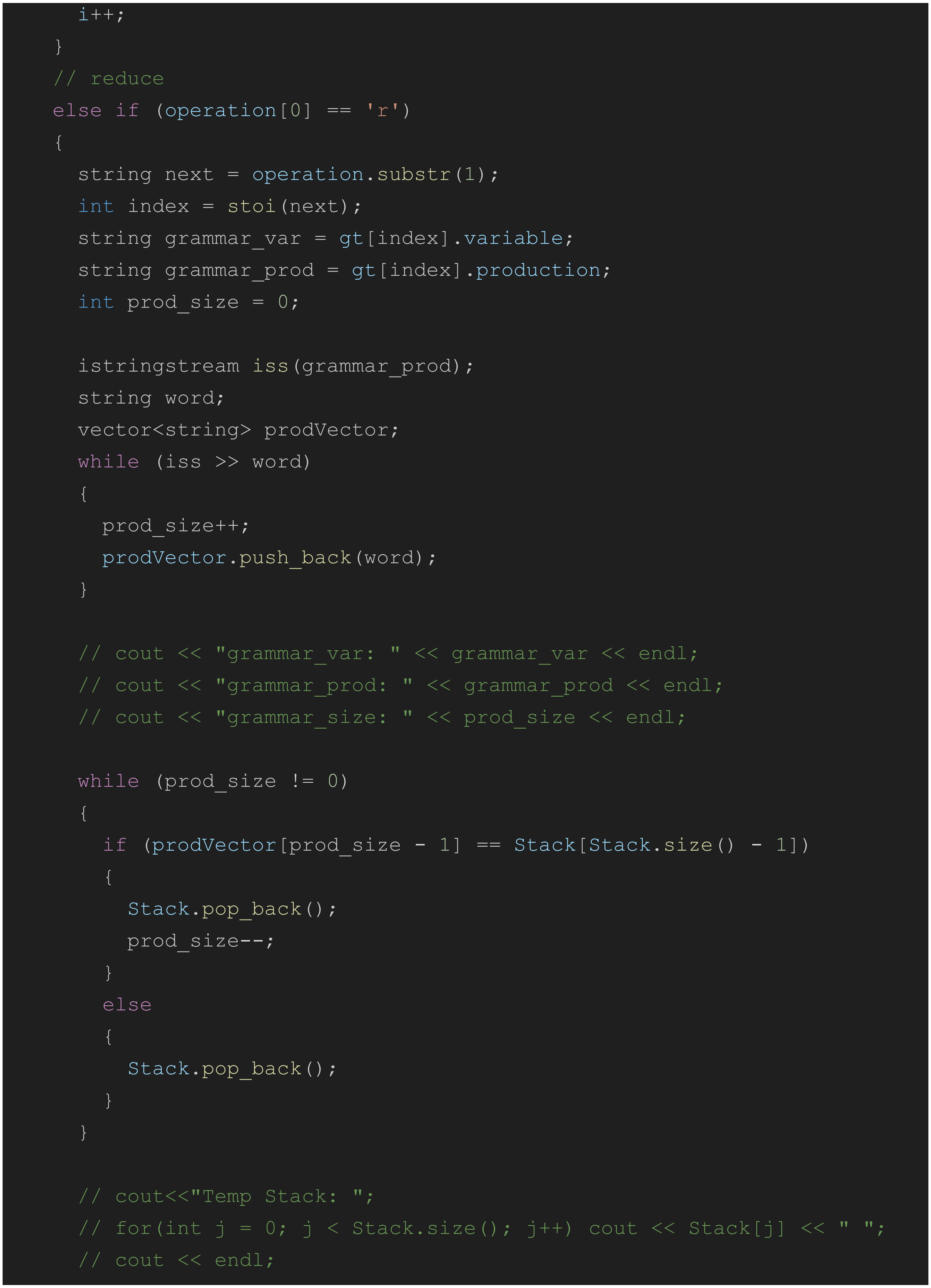
```
for (int j = 0; j < ColumnWithIndex.size(); j++)
{
  (row.columns).push_back("-");
}

for (int j = 0; j < (States[i].var).size(); j++)
{
  string curr_prod = States[i].prod[j];
  string curr_var = States[i].var[j];

  vector<string> prod_vector;
  int dotPosition = getDotPosition(curr_prod, prod_vector);

  // reduce
  if (dotPosition == prod_vector.size() - 1)
  {
    string prod = curr_prod.substr(0, curr_prod.length() - 2);

    int grammar_index = getGrammarIndex(gt, curr_var, prod);

    for (int k = 0; k < followTable.size(); k++)
    {
      if (followTable[k].row_name == curr_var)
      {
        for (int l = 0; l < (followTable[k].columns).size(); l++)
        {
          string followers = followTable[k].columns[l];
          int colNum = ColumnWithIndex.at(followers);

          if (i == 1 && followers == "$")
          {
            row.columns[colNum] = "A";
            continue;
          }
          row.columns[colNum] = "r" + to_string(grammar_index);
        }
        break;
      }
    }
```



```
        continue;
      }

      // shift
      string nextToDot = prod_vector[dotPosition + 1];

      int next_index = States[i].next[j];

      if (nextToDot[0] >= 'A' && nextToDot[0] <= 'Z')
      {
        int colNum = ColumnWithIndex.at(nextToDot);

        row.columns[colNum] = to_string(next_index);
      }
      else
      {
        int colNum = ColumnWithIndex.at(nextToDot);

        row.columns[colNum] = "s" + to_string(next_index);
      }
    }

    SLRtable.push_back(row);
  }
}

string getOperation(vector<table> SLRtable, string TOS, int columnIndex)
{
  string operation = "";
  // cout<<"in getOP"<<endl;
  // cout<<"TOS="<<TOS<<", index="<<columnIndex<<endl;

  for (int i = 0; i < SLRtable.size(); i++)
  {
    if (SLRtable[i].row_name == TOS)
    {
      operation = SLRtable[i].columns[columnIndex];
      break;
    }
  }

  return operation;
}

void printOut(vector<string> Stack, vector<string> input_chars, int i,
FILE *q)
{
  int spaceSize = (input_chars.size() * 2) - Stack.size();
  string space = "";

  for (int i = 0; i < spaceSize; i++)
    space += " ";

  for (int j = 0; j < Stack.size(); j++)
    fprintf(q, "%s", Stack[j].c_str());

  fprintf(q, "%s", space.c_str());

  for (int j = i; j < input_chars.size(); j++)
    fprintf(q, "%s", input_chars[j].c_str());

  fprintf(q, "\n");
}

void parseString(string inputString, vector<table> SLRtable,
vector<grammar_table> gt, unordered_map<string, int> ColumnWithIndex)
{
  inputString += " $";
  istringstream iss(inputString);
  string word;
  vector<string> inputVector;
  vector<string> Stack;

  // int index = 0;

  Stack.push_back("0");
  int stackSize = Stack.size();
  string TOS = "0";
```

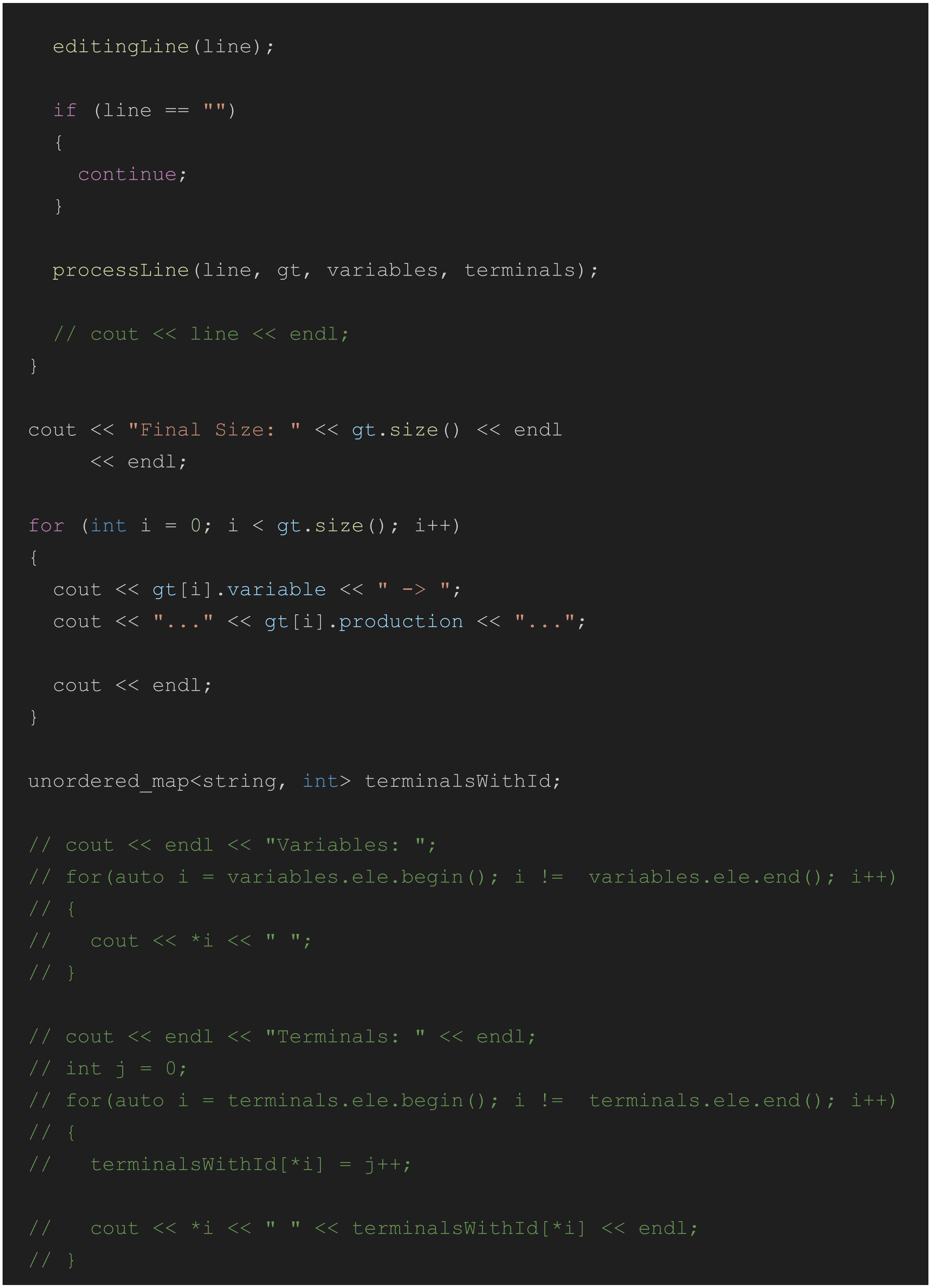


```
// int dotPosition = 0;
while (iss >> word)
{
  inputVector.push_back(word);
}

int i = 0;

FILE *q = fopen("output.txt", "w");
int spaceSize = (inputVector.size() * 2) - Stack.size();
string space = "";

for (int i = 0; i < spaceSize; i++)
  space += " ";
fprintf(q, "Stack%sInput\n", space.c_str());

while (stackSize != 0)
{
  printOut(Stack, inputVector, i, q);

  cout << "Stack: ";
  for (int j = 0; j < Stack.size(); j++)
    cout << Stack[j] << " ";
  cout << endl;

  string curr_input = inputVector[i];

  int columnIndex = ColumnWithIndex.at(curr_input);

  string operation = getOperation(SLRtable, TOS, columnIndex);

  cout << "Operation = " << operation << endl;

  // shift
  if (operation[0] == 's')
  {
    string next = operation.substr(1);

    Stack.push_back(curr_input);
    Stack.push_back(next);
```

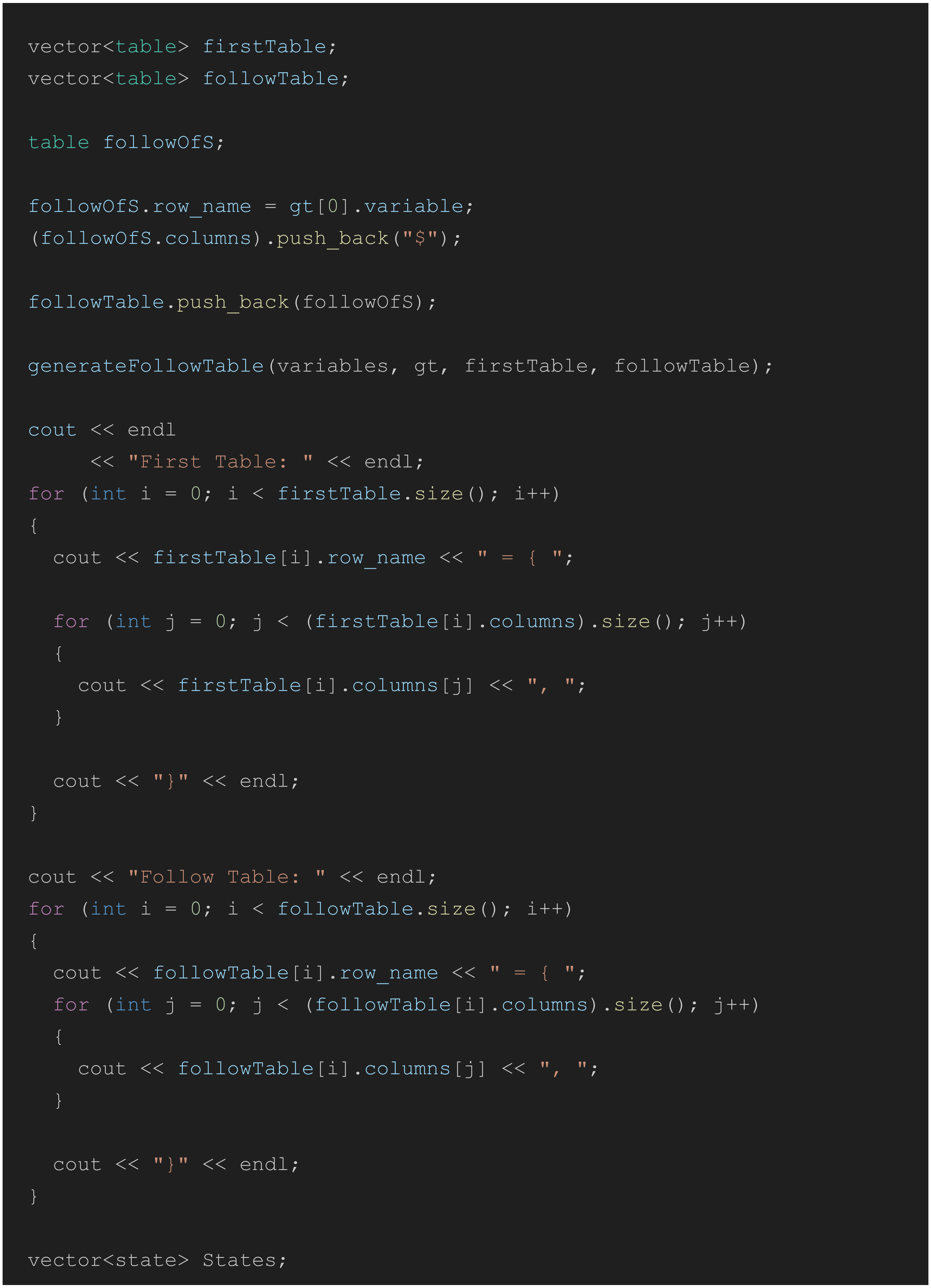


```
      i++;
    }
    // reduce
    else if (operation[0] == 'r')
    {
      string next = operation.substr(1);
      int index = stoi(next);
      string grammar_var = gt[index].variable;
      string grammar_prod = gt[index].production;
      int prod_size = 0;

      istringstream iss(grammar_prod);
      string word;
      vector<string> prodVector;
      while (iss >> word)
      {
        prod_size++;
        prodVector.push_back(word);
      }

      // cout << "grammar_var: " << grammar_var << endl;
      // cout << "grammar_prod: " << grammar_prod << endl;
      // cout << "grammar_size: " << prod_size << endl;

      while (prod_size != 0)
      {
        if (prodVector[prod_size - 1] == Stack[Stack.size() - 1])
        {
          Stack.pop_back();
          prod_size--;
        }
        else
        {
          Stack.pop_back();
        }
      }

      // cout<<"Temp Stack: ";
      // for(int j = 0; j < Stack.size(); j++) cout << Stack[j] << " ";
      // cout << endl;
```

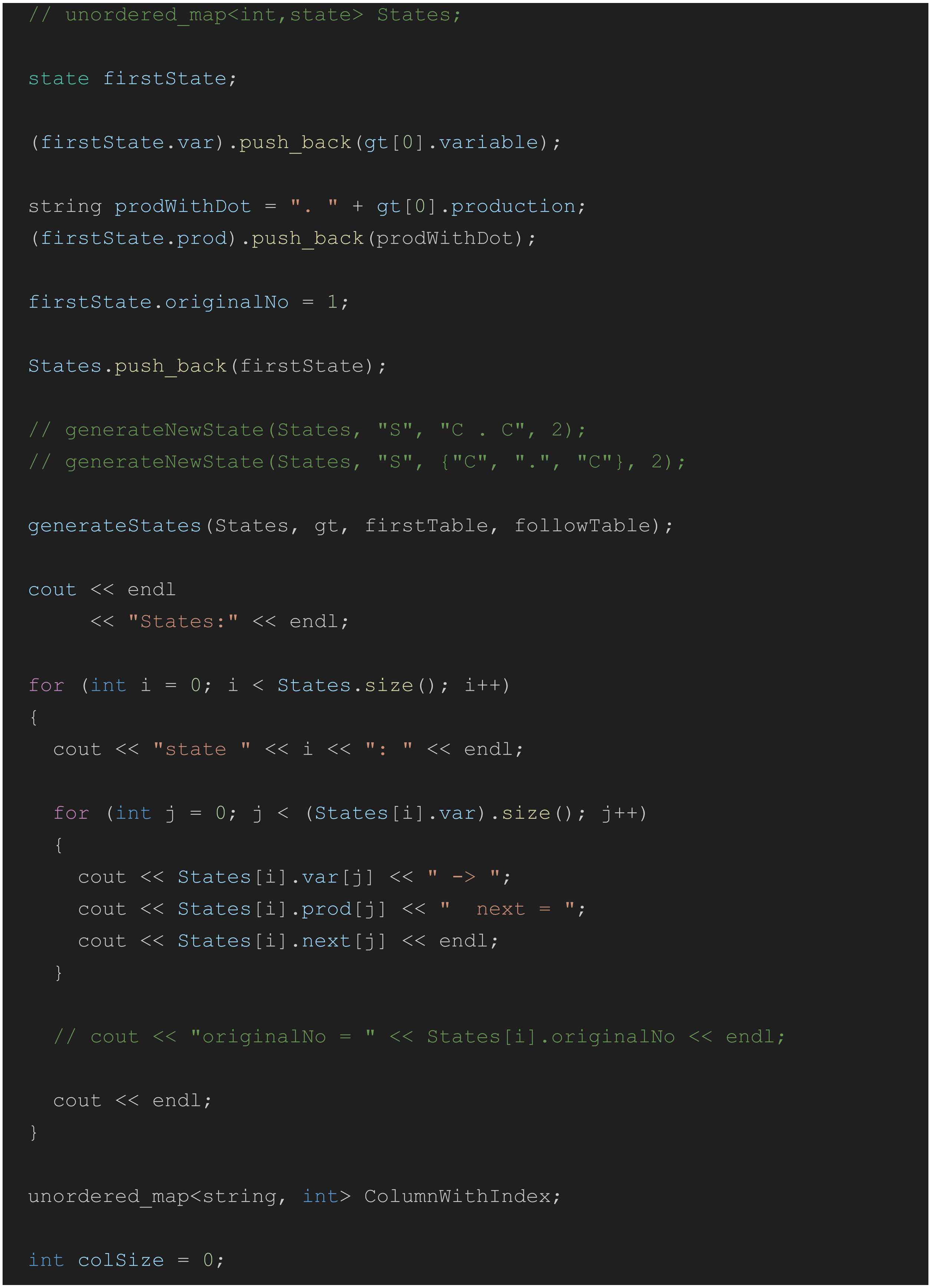
```
      string temp_TOS = Stack[Stack.size() - 1];

      int columnIndex = ColumnWithIndex.at(grammar_var);

      string temp_op = getOperation(SLRtable, temp_TOS, columnIndex);
      Stack.push_back(grammar_var);
      Stack.push_back(temp_op);
    }
    else if (operation[0] == 'A')
    {
      cout << "String belongs to grammar" << endl;
      break;
    }
    else
    {
      cout << "Error" << endl;
      break;
    }

    TOS = Stack[Stack.size() - 1];
  }
}

int main()
{
  FILE *grammar = fopen("grammar.txt", "r");
  char input[100];

  vector<grammar_table> gt;

  elements variables;
  elements terminals;

  variables.count = 0;
  terminals.count = 0;

  while (fgets(input, 100, grammar) != NULL)
  {
    string line = input;
```



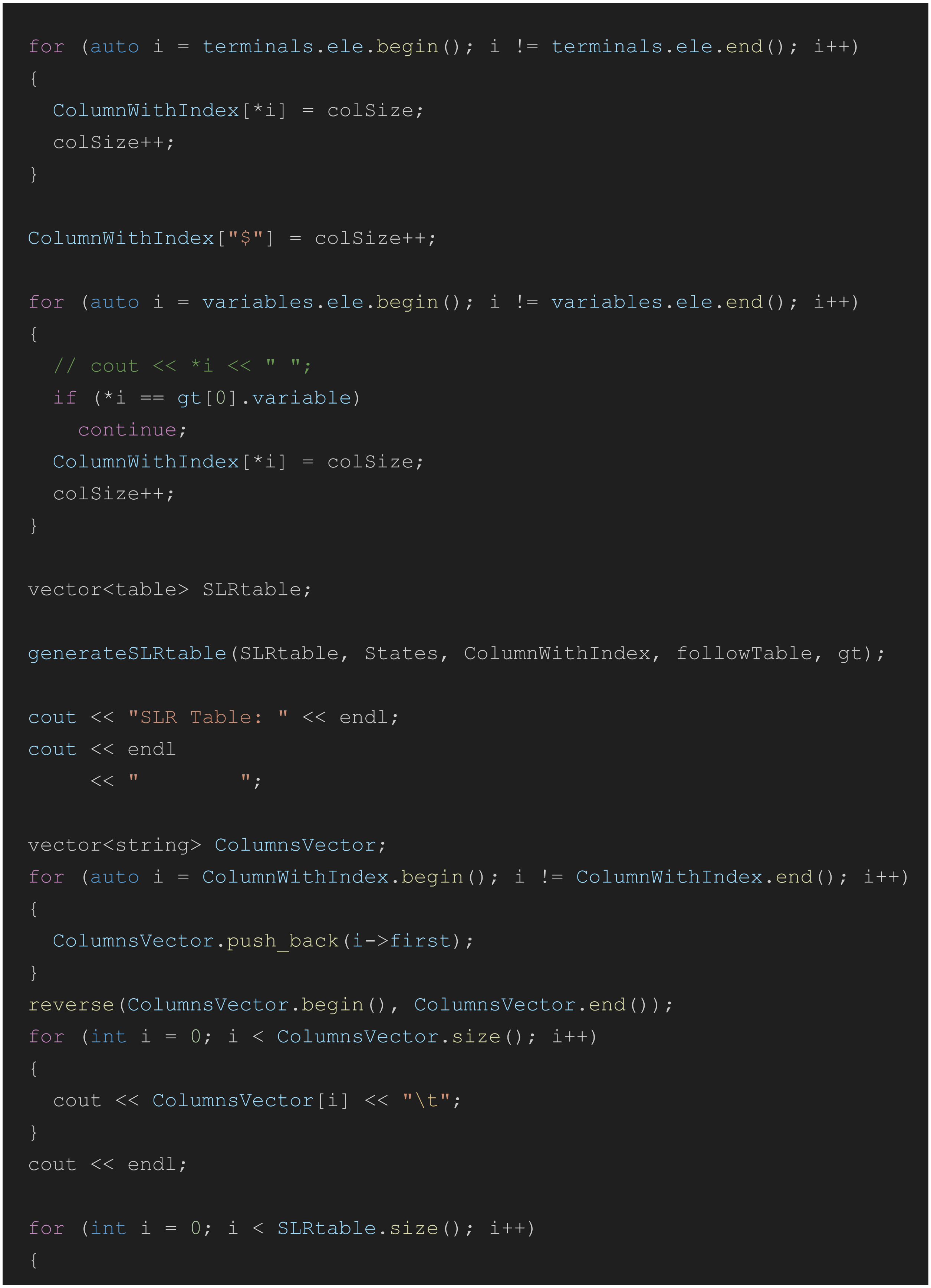
```
  editingLine(line);

  if (line == "")
  {
    continue;
  }

  processLine(line, gt, variables, terminals);

  // cout << line << endl;
}

cout << "Final Size: " << gt.size() << endl
     << endl;

for (int i = 0; i < gt.size(); i++)
{
  cout << gt[i].variable << " -> ";
  cout << "..." << gt[i].production << "...";

  cout << endl;
}

unordered_map<string, int> terminalsWithId;

// cout << endl << "Variables: ";
// for(auto i = variables.ele.begin(); i !=  variables.ele.end(); i++)
// {
//   cout << *i << " ";
// }

// cout << endl << "Terminals: " << endl;
// int j = 0;
// for(auto i = terminals.ele.begin(); i !=  terminals.ele.end(); i++)
// {
//   terminalsWithId[*i] = j++;

//   cout << *i << " " << terminalsWithId[*i] << endl;
// }
```



```cpp
vector<table> firstTable;
vector<table> followTable;

table followOfS;

followOfS.row_name = gt[0].variable;
(followOfS.columns).push_back("$");

followTable.push_back(followOfS);

generateFollowTable(variables, gt, firstTable, followTable);

cout << endl
     << "First Table: " << endl;
for (int i = 0; i < firstTable.size(); i++)
{
  cout << firstTable[i].row_name << " = { ";

  for (int j = 0; j < (firstTable[i].columns).size(); j++)
  {
    cout << firstTable[i].columns[j] << ", ";
  }

  cout << "}" << endl;
}

cout << "Follow Table: " << endl;
for (int i = 0; i < followTable.size(); i++)
{
  cout << followTable[i].row_name << " = { ";
  for (int j = 0; j < (followTable[i].columns).size(); j++)
  {
    cout << followTable[i].columns[j] << ", ";
  }

  cout << "}" << endl;
}

vector<state> States;
```

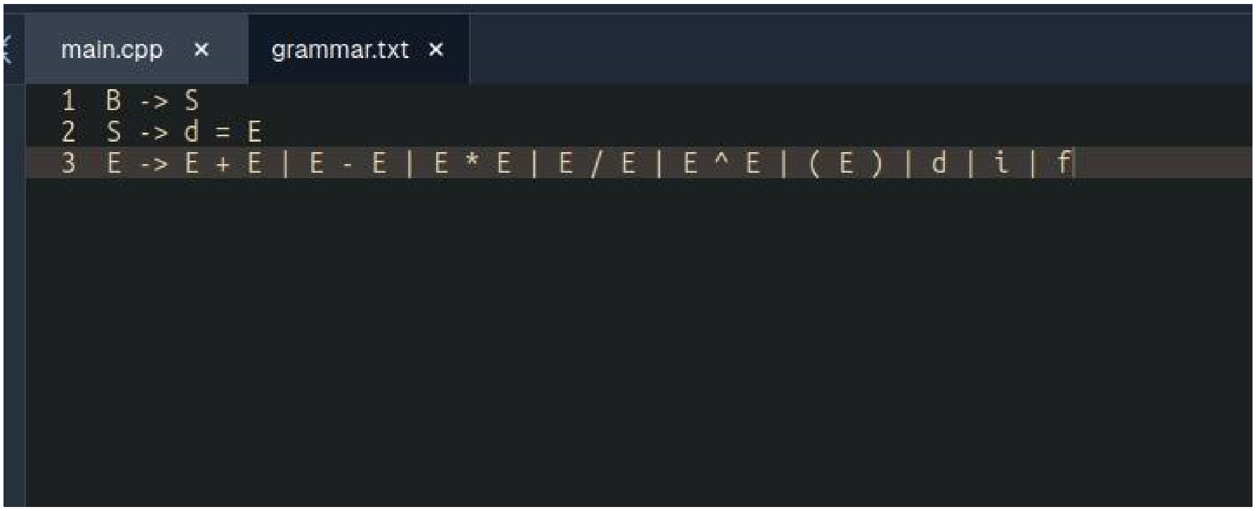


```
// unordered_map<int,state> States;

state firstState;

(firstState.var).push_back(gt[0].variable);

string prodWithDot = ". " + gt[0].production;
(firstState.prod).push_back(prodWithDot);

firstState.originalNo = 1;

States.push_back(firstState);

// generateNewState(States, "S", "C . C", 2);
// generateNewState(States, "S", {"C", ".", "C"}, 2);

generateStates(States, gt, firstTable, followTable);

cout << endl
     << "States:" << endl;

for (int i = 0; i < States.size(); i++)
{
  cout << "state " << i << ": " << endl;

  for (int j = 0; j < (States[i].var).size(); j++)
  {
    cout << States[i].var[j] << " -> ";
    cout << States[i].prod[j] << "  next = ";
    cout << States[i].next[j] << endl;
  }

  // cout << "originalNo = " << States[i].originalNo << endl;

  cout << endl;
}

unordered_map<string, int> ColumnWithIndex;

int colSize = 0;
```



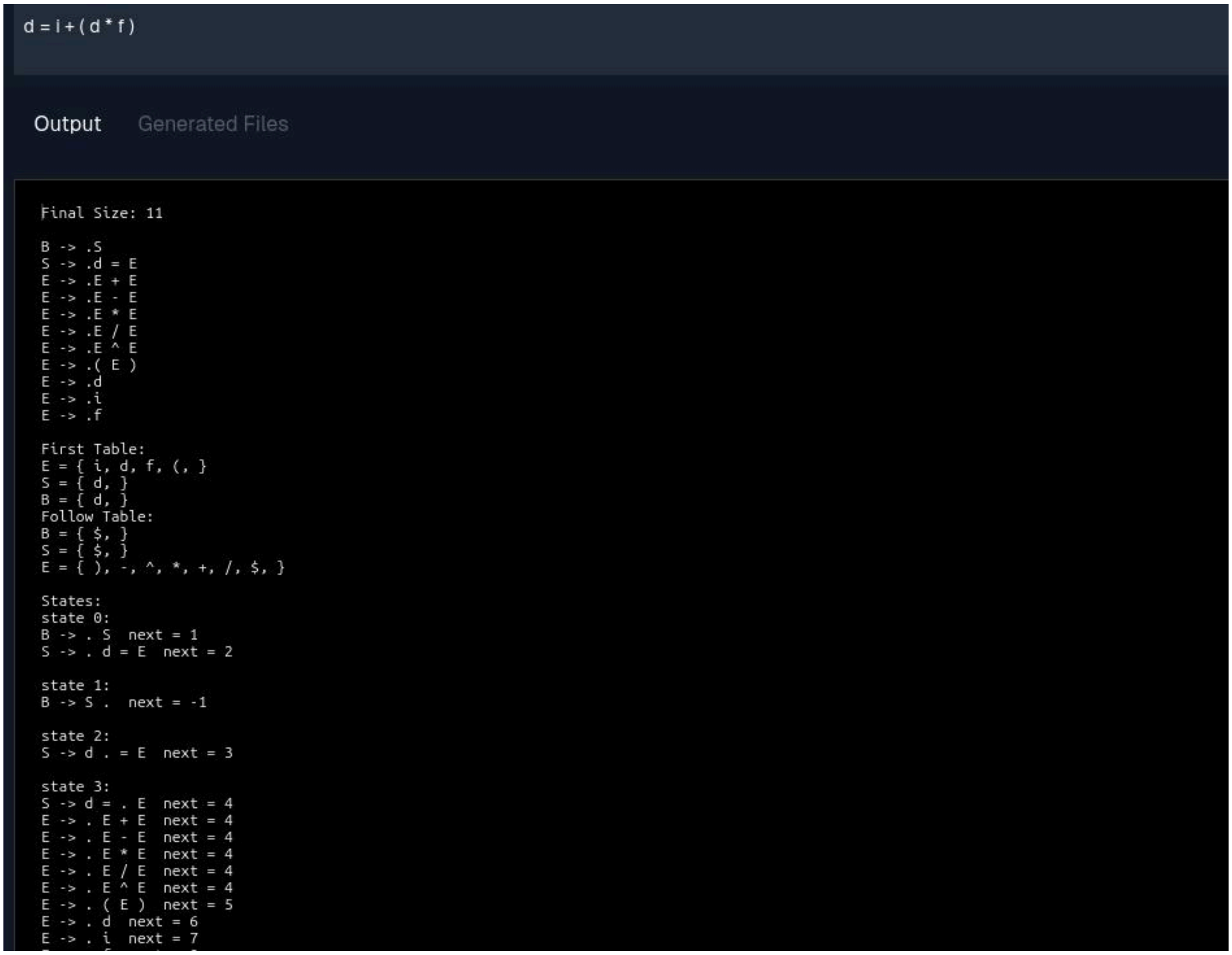
```
for (auto i = terminals.ele.begin(); i != terminals.ele.end(); i++)
{
  ColumnWithIndex[*i] = colSize;
  colSize++;
}

ColumnWithIndex["$"] = colSize++;

for (auto i = variables.ele.begin(); i != variables.ele.end(); i++)
{
  // cout << *i << " ";
  if (*i == gt[0].variable)
    continue;
  ColumnWithIndex[*i] = colSize;
  colSize++;
}

vector<table> SLRtable;

generateSLRtable(SLRtable, States, ColumnWithIndex, followTable, gt);

cout << "SLR Table: " << endl;
cout << endl
     << "        ";

vector<string> ColumnsVector;
for (auto i = ColumnWithIndex.begin(); i != ColumnWithIndex.end(); i++)
{
  ColumnsVector.push_back(i->first);
}
reverse(ColumnsVector.begin(), ColumnsVector.end());
for (int i = 0; i < ColumnsVector.size(); i++)
{
  cout << ColumnsVector[i] << "\t";
}
cout << endl;

for (int i = 0; i < SLRtable.size(); i++)
{
    cout << "I" << SLRtable[i].row_name << "\t= { ";
    for (int j = 0; j < (SLRtable[i].columns).size(); j++)
    {
      cout << SLRtable[i].columns[j] << "\t";
    }

    cout << "}" << endl;
  }

  string inputString;
  cout << endl
       << "Enter input string\n";
  getline(cin, inputString);

  cout << endl
       << "Parsing the given String:\n";

  parseString(inputString, SLRtable, gt, ColumnWithIndex);

  return 0;
}
```

Input:



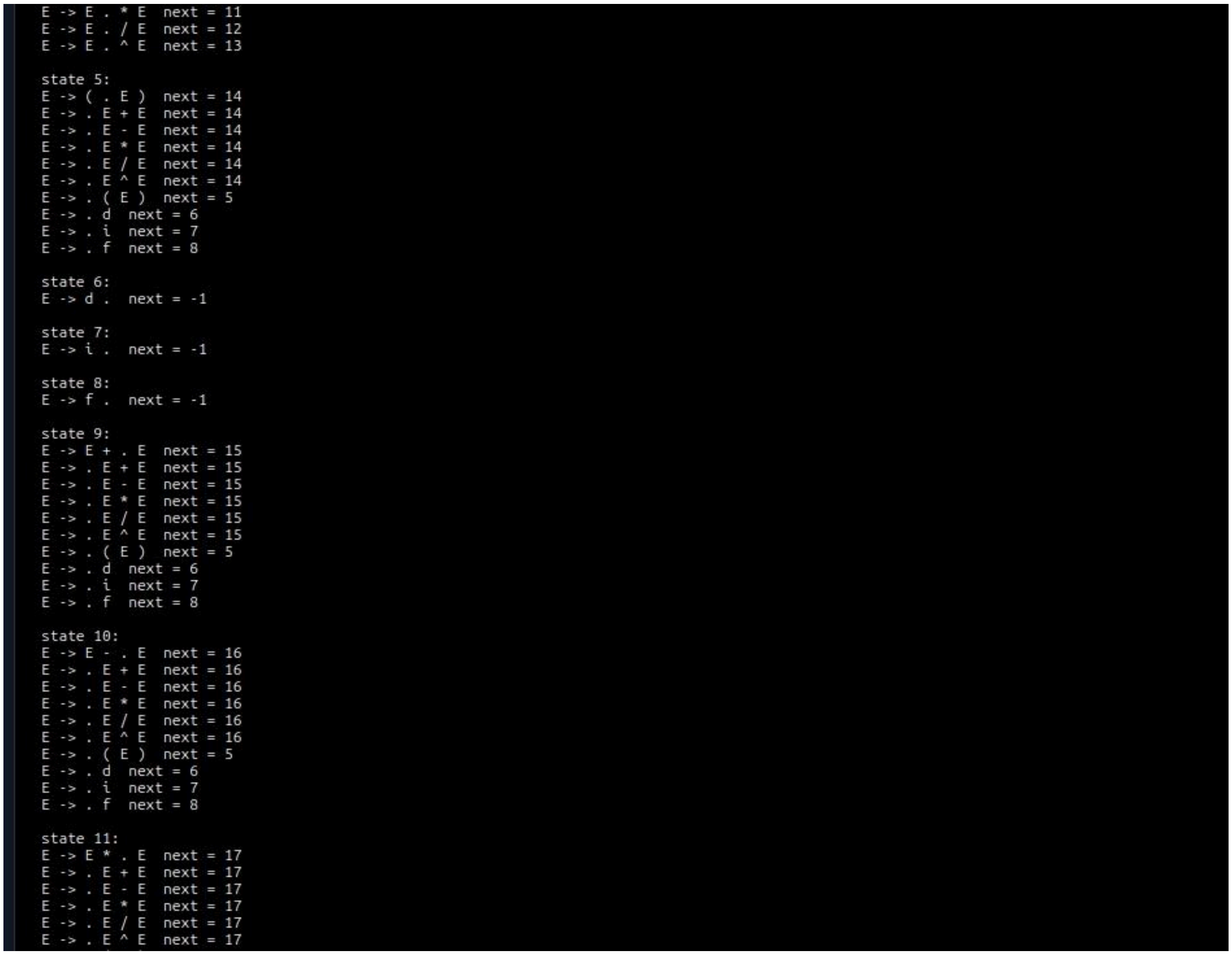
```
B -> S
S -> d = E
E -> E + E | E - E | E * E | E / E | E ^ E | ( E ) | d | i | f
```

Output:

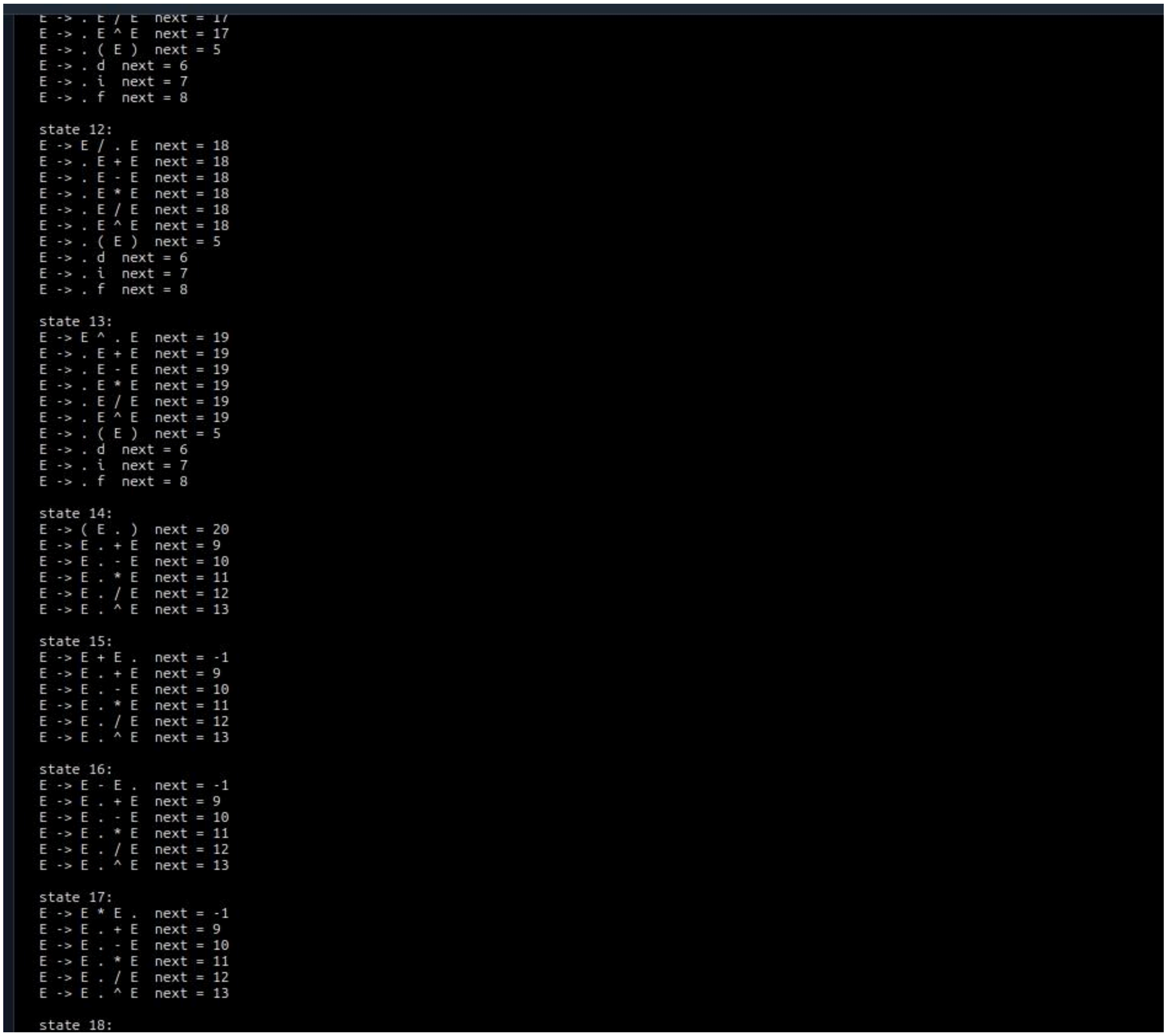


```
d = i + ( d * f )
```

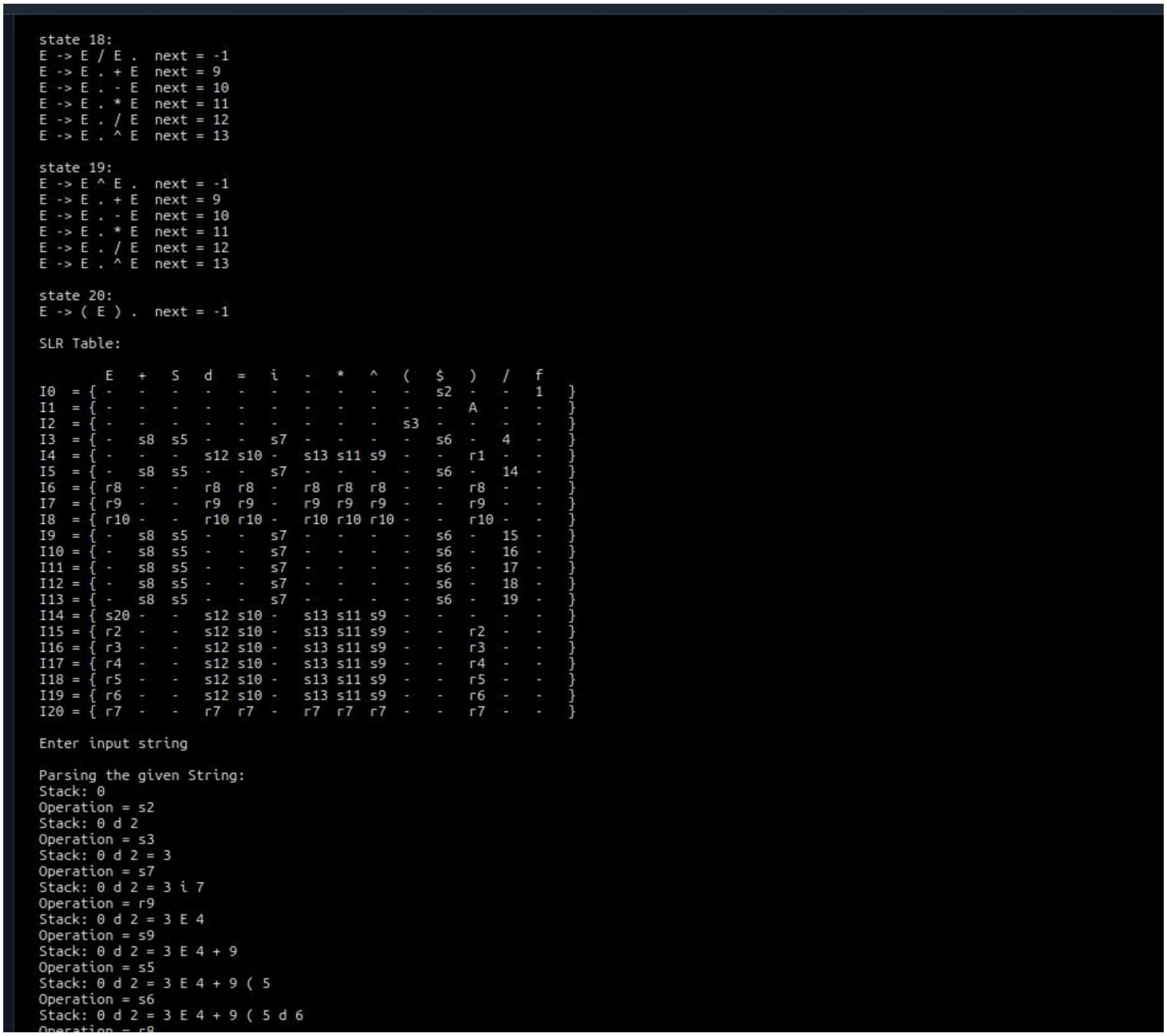
```
Final Size: 11

B -> .S
S -> .d = E
E -> .E + E
E -> .E - E
E -> .E * E
E -> .E / E
E -> .E ^ E
E -> .( E )
E -> .d
E -> .i
E -> .f

First Table:
E = { i, d, f, (, }
S = { d, }
B = { d, }
Follow Table:
B = { $, }
S = { $, }
E = { ), -, ^, *, +, /, $, }

States:
state 0:
B -> . S  next = 1
S -> . d = E  next = 2

state 1:
B -> S .  next = -1

state 2:
S -> d . = E  next = 3

state 3:
S -> d = . E  next = 4
E -> . E + E  next = 4
E -> . E - E  next = 4
E -> . E * E  next = 4
E -> . E / E  next = 4
E -> . E ^ E  next = 4
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8
```



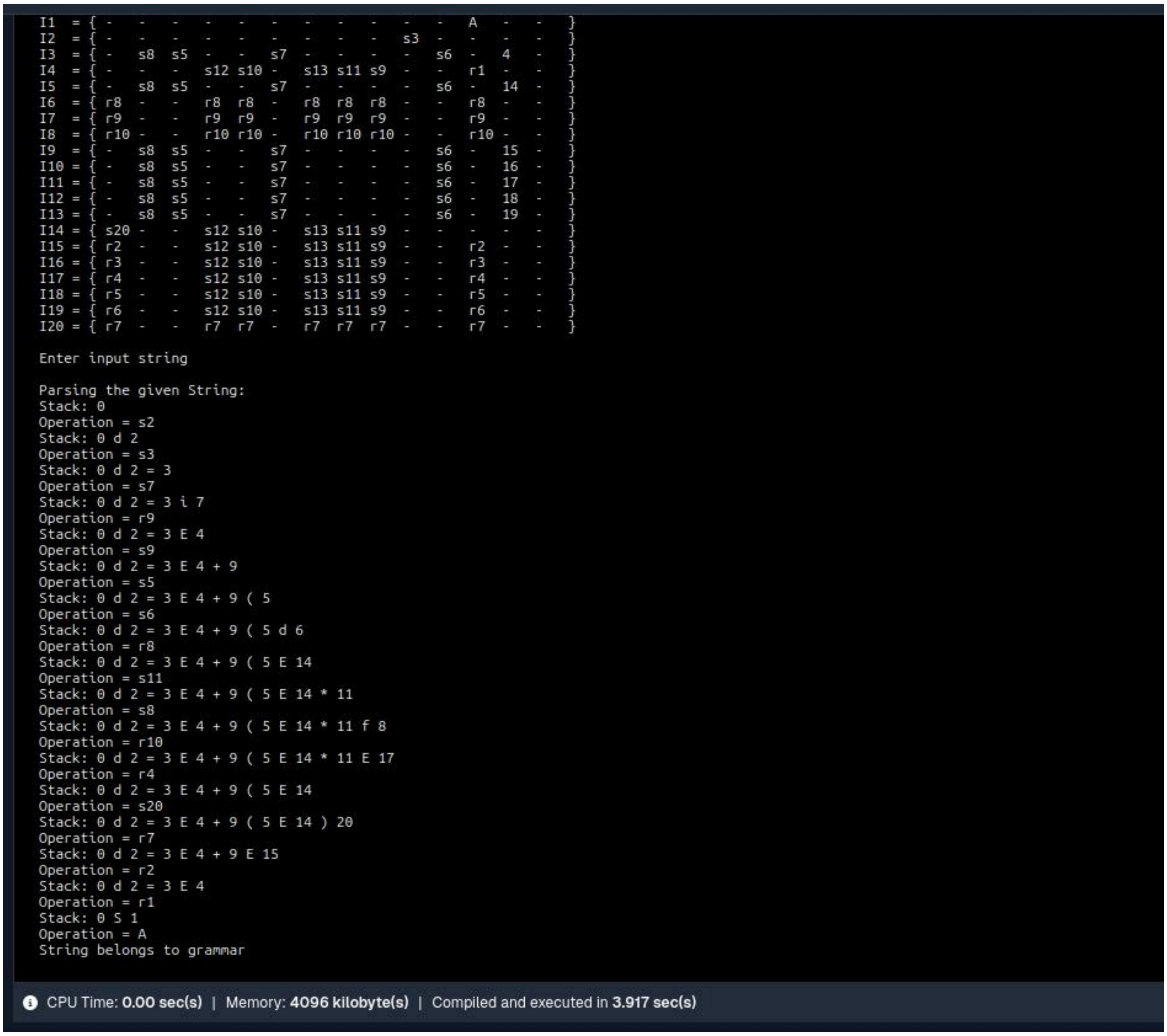
```
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 5:
E -> ( . E )  next = 14
E -> . E + E  next = 14
E -> . E - E  next = 14
E -> . E * E  next = 14
E -> . E / E  next = 14
E -> . E ^ E  next = 14
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 6:
E -> d .  next = -1

state 7:
E -> i .  next = -1

state 8:
E -> f .  next = -1

state 9:
E -> E + . E  next = 15
E -> . E + E  next = 15
E -> . E - E  next = 15
E -> . E * E  next = 15
E -> . E / E  next = 15
E -> . E ^ E  next = 15
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 10:
E -> E - . E  next = 16
E -> . E + E  next = 16
E -> . E - E  next = 16
E -> . E * E  next = 16
E -> . E / E  next = 16
E -> . E ^ E  next = 16
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 11:
E -> E * . E  next = 17
E -> . E + E  next = 17
E -> . E - E  next = 17
E -> . E * E  next = 17
E -> . E / E  next = 17
E -> . E ^ E  next = 17
```



```
E -> . E / E  next = 17
E -> . E ^ E  next = 17
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 12:
E -> E / . E  next = 18
E -> . E + E  next = 18
E -> . E - E  next = 18
E -> . E * E  next = 18
E -> . E / E  next = 18
E -> . E ^ E  next = 18
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 13:
E -> E ^ . E  next = 19
E -> . E + E  next = 19
E -> . E - E  next = 19
E -> . E * E  next = 19
E -> . E / E  next = 19
E -> . E ^ E  next = 19
E -> . ( E )  next = 5
E -> . d  next = 6
E -> . i  next = 7
E -> . f  next = 8

state 14:
E -> ( E . )  next = 20
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 15:
E -> E + E .  next = -1
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 16:
E -> E - E .  next = -1
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 17:
E -> E * E .  next = -1
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 18:
```



```
state 18:
E -> E / E .  next = -1
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 19:
E -> E ^ E .  next = -1
E -> E . + E  next = 9
E -> E . - E  next = 10
E -> E . * E  next = 11
E -> E . / E  next = 12
E -> E . ^ E  next = 13

state 20:
E -> ( E ) .  next = -1

SLR Table:

        E   +   S   d   =   i   -   *   ^   (   $   )   /   f
I0  = { -   -   -   -   -   -   -   -   -   -   s2  -   -   1   }
I1  = { -   -   -   -   -   -   -   -   -   -   -   A   -   -   }
I2  = { -   -   -   -   -   -   -   -   -   s3  -   -   -   -   }
I3  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   4   -   }
I4  = { -   -   -   s12 s10 -   s13 s11 s9  -   -   r1  -   -   }
I5  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   14  -   }
I6  = { r8  -   -   r8  r8  -   r8  r8  r8  -   -   r8  -   -   }
I7  = { r9  -   -   r9  r9  -   r9  r9  r9  -   -   r9  -   -   }
I8  = { r10 -   -   r10 r10 -   r10 r10 r10 -   -   r10 -   -   }
I9  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   15  -   }
I10 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   16  -   }
I11 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   17  -   }
I12 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   18  -   }
I13 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   19  -   }
I14 = { s20 -   -   s12 s10 -   s13 s11 s9  -   -   -   -   -   }
I15 = { r2  -   -   s12 s10 -   s13 s11 s9  -   -   r2  -   -   }
I16 = { r3  -   -   s12 s10 -   s13 s11 s9  -   -   r3  -   -   }
I17 = { r4  -   -   s12 s10 -   s13 s11 s9  -   -   r4  -   -   }
I18 = { r5  -   -   s12 s10 -   s13 s11 s9  -   -   r5  -   -   }
I19 = { r6  -   -   s12 s10 -   s13 s11 s9  -   -   r6  -   -   }
I20 = { r7  -   -   r7  r7  -   r7  r7  r7  -   -   r7  -   -   }

Enter input string

Parsing the given String:
Stack: 0
Operation = s2
Stack: 0 d 2
Operation = s3
Stack: 0 d 2 = 3
Operation = s7
Stack: 0 d 2 = 3 i 7
Operation = r9
Stack: 0 d 2 = 3 E 4
Operation = s9
Stack: 0 d 2 = 3 E 4 + 9
Operation = s5
Stack: 0 d 2 = 3 E 4 + 9 ( 5
Operation = s6
Stack: 0 d 2 = 3 E 4 + 9 ( 5 d 6
Operation = r8
```



```
I1  = { -   -   -   -   -   -   -   -   -   -   -   A   -   -   }
I2  = { -   -   -   -   -   -   -   -   -   s3  -   -   -   -   }
I3  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   4   -   }
I4  = { -   -   -   s12 s10 -   s13 s11 s9  -   -   r1  -   -   }
I5  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   14  -   }
I6  = { r8  -   -   r8  r8  -   r8  r8  r8  -   -   r8  -   -   }
I7  = { r9  -   -   r9  r9  -   r9  r9  r9  -   -   r9  -   -   }
I8  = { r10 -   -   r10 r10 -   r10 r10 r10 -   -   r10 -   -   }
I9  = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   15  -   }
I10 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   16  -   }
I11 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   17  -   }
I12 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   18  -   }
I13 = { -   s8  s5  -   -   s7  -   -   -   -   s6  -   19  -   }
I14 = { s20 -   -   s12 s10 -   s13 s11 s9  -   -   -   -   -   }
I15 = { r2  -   -   s12 s10 -   s13 s11 s9  -   -   r2  -   -   }
I16 = { r3  -   -   s12 s10 -   s13 s11 s9  -   -   r3  -   -   }
I17 = { r4  -   -   s12 s10 -   s13 s11 s9  -   -   r4  -   -   }
I18 = { r5  -   -   s12 s10 -   s13 s11 s9  -   -   r5  -   -   }
I19 = { r6  -   -   s12 s10 -   s13 s11 s9  -   -   r6  -   -   }
I20 = { r7  -   -   r7  r7  -   r7  r7  r7  -   -   r7  -   -   }

Enter input string

Parsing the given String:
Stack: 0
Operation = s2
Stack: 0 d 2
Operation = s3
Stack: 0 d 2 = 3
Operation = s7
Stack: 0 d 2 = 3 i 7
Operation = r9
Stack: 0 d 2 = 3 E 4
Operation = s9
Stack: 0 d 2 = 3 E 4 + 9
Operation = s5
Stack: 0 d 2 = 3 E 4 + 9 ( 5
Operation = s6
Stack: 0 d 2 = 3 E 4 + 9 ( 5 d 6
Operation = r8
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14
Operation = s11
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14 * 11
Operation = s8
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14 * 11 f 8
Operation = r10
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14 * 11 E 17
Operation = r4
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14
Operation = s20
Stack: 0 d 2 = 3 E 4 + 9 ( 5 E 14 ) 20
Operation = r7
Stack: 0 d 2 = 3 E 4 + 9 E 15
Operation = r2
Stack: 0 d 2 = 3 E 4
Operation = r1
Stack: 0 S 1
Operation = A
String belongs to grammar
```

CPU Time: **0.00 sec(s)** | Memory: **4096 kilobyte(s)** | Compiled and executed in **3.917 sec(s)**