```
#include "OURFUNCTIONS.h"
#include <iostream>
#include <string>
#include <fstream>
#include <cctype>
#include <sstream>

using namespace std;

void leftTrim(string& current)
{
     size_t pos = current.find_first_not_of(" ");
     current = current.substr(pos);
}

void rightTrim(string& current)
{
     size_t pos = current.find_last_not_of(" ");
     current = current.substr(0, pos + 1);
}

void cleanup(string filename)
{
     cout << "Cleaning up text file..." << endl;
     fstream original;
     original.open(filename, ios::in);
     string token;

     ofstream newFile("finalp2.txt");

     if (original.is_open())
     {
          getline(original, token);
          bool isComment = false;
          while (original)
          {
               // If line is not blank, perform the following
               if (token.size() > 0 && token.find_first_not_of(" ") != string::npos)
               {
                    leftTrim(token);
                    rightTrim(token);
                    size_t semicolon = token.find(';');
                    // If line contains semicolon, create substring ending
                    // at that ';' position
                    if (semicolon != string::npos)
                    {
                         token = token.substr(0, semicolon + 1);
                    }
                    // If line begins with a comment or is the ending
                    // of a comment, remove those lines
                    if (token.front() == '/' || token.back() == '/')
                    {
                         isComment = true;
                    }
                    // If line is not a comment, perform the following
                    if (!isComment)
                    {
                         // Iterate through string
                         for (size_t i = 0; i < token.size(); ++i)
                         {
                              // If literal string detected, perform the following
                              if (token[i] == '"')
                              {
                                   int newPos = token.find_last_of('"');
                                   i = newPos;
                              }
```

```
                              // If whitespace, perform the following
                              else if (token[i] == ' ')
                              {
                                   if (token[i + 1] == ' ')
                                   {
                                        token.erase(i, 1);
                                        --i;
                                   }

                              }
                              // If any arithmetic operators, perform the following
                              else if (token[i] == '+' || token[i] == '-'
                                   || token[i] == '*' ||
                                   token[i] == '=' || token[i] == '('
                                   || token[i] == ')' || token[i] == ',')
                              {
                                   if (token[i - 1] != ' ')
                                   {
                                        token.insert(i, " ");
                                        ++i;
                                   }
                                   if (token[i + 1] != ' ')
                                   {
                                        token.insert(i + 1, " ");
                                   }
                              }
                              else if (token[i] == ';')
                              {
                                   if (token[i - 1] != ' ')
                                   {
                                        token.insert(i, " ");
                                        ++i;
                                   }
                              }
                         }
                         // Add token to newFile
                         newFile << token;
                         // If buffer has remaining text, add new line
                         if (original)
                         {
                              newFile << endl;
                         }
                    }
               }
               // Reset isComment flag
               isComment = false;
               // Move onto next line in buffer
               getline(original, token);
          }

     }
     newFile << "$";
     cout << "Cleanup successful!" << endl << endl;
     newFile.close();
}

void errorCheck(string newProgram, bool& errorFound)
{
     cout << "Checking for errors..." << endl;
     fstream programInput;
     programInput.open(newProgram, ios::in);
     string token;
     bool programFound = false;
     bool varFound = false;
     bool beginFound = false;
     bool endFound = false;
```

```
bool integerFound = false;
bool missingSemi = false;
bool missingComma = false;
bool missingPeriod = false;
bool missingLeftParen = false;
bool missingRightParen = false;

if (programInput.is_open())
{
     while (programInput)
     {
          getline(programInput, token);
          checkReserved(programFound, varFound, beginFound, endFound,
               integerFound, token);
          if (token.find("var") == string::npos && token.find("begin") == string::npos
               && token.find("end.") == string::npos && token.find("$") == string::npos)
          {
               if (token.back() != ';')
               {
                    missingSemi = true;
               }
               if (token.find("integer") != string::npos)
               {
                    if (token.find(":") == string::npos)
                    {
                         missingComma = true;
                    }
               }
               if (token.find("end") != string::npos)
               {
                    if (token.find(".") == string::npos)
                    {
                         missingPeriod = true;
                    }
               }
               if (token.find("(") != string::npos)
               {
                    if (token.find(")") == string::npos)
                    {
                         missingRightParen = true;
                    }

               }
               if (token.find(")") != string::npos)
               {
                    if (token.find("(") == string::npos)
                    {
                         missingLeftParen = true;
                    }

               }
          }
     }
     if (!programFound || !varFound || !beginFound || !endFound ||
          !integerFound || missingSemi || missingComma || missingPeriod ||
               missingLeftParen || missingRightParen)
     {
          errorFound = true;
          string errorType;
          if (!programFound) errorType = "program";
          else if (!varFound) errorType = "var";
          else if (!beginFound) errorType = "begin";
          else if (!endFound) errorType = "end.";
          else if (!integerFound) errorType = "integer";
          else if (missingSemi) errorType = ";";
          else if (missingComma) errorType = ",";
```

```
                else if (missingPeriod) errorType = ".";
                else if (missingLeftParen) errorType = "(";
                else if (missingRightParen) errorType = ")";
                cerr << "ERROR: " << errorType << " is expected";
                if (!integerFound)
                {
                     cerr << ", unknown identifier if variable is not defined";
                }
                cout << endl << endl;
           }
      }
      if (!errorFound)
      {
           cout << "Success! No errors found." << endl << endl;
      }

      programInput.close();
}

void checkReserved(bool& programFound, bool& varFound, bool& beginFound,
      bool& endFound, bool& integerFound, string str)
{
      size_t found = str.find("program");
      if (found != string::npos)
      {
           programFound = true;
      }
      found = str.find("var");
      if (found != string::npos)
      {
           varFound = true;
      }
      found = str.find("begin");
      if (found != string::npos)
      {
           beginFound = true;
      }
      found = str.find("end.");
      if (found != string::npos)
      {
           endFound = true;
      }
      found = str.find("integer");
      if (found != string::npos)
      {
           integerFound = true;

      }
}

void addToStack(stack<string>& myStack, string entry)
{
      stack<string> reverseStack;
      istringstream toReverse(entry);
      do {
           string rev;
           toReverse >> rev;
           reverseStack.push(rev);

      } while (toReverse);

      while (!reverseStack.empty())
      {
           myStack.push(reverseStack.top());
           reverseStack.pop();
      }
```

```
}

void getCol(string a, int& col)
{
     if (a == "$") col = 0;
     else if (a == ";") col = 1;
     else if (a == ",") col = 2;
     else if (a == "+") col = 3;
     else if (a == "-") col = 4;
     else if (a == "=") col = 5;
     else if (a == ":") col = 6;
     else if (a == "/") col = 7;
     else if (a == "*") col = 8;
     else if (a == "(") col = 9;
     else if (a == ")") col = 10;
     else if (a == "p") col = 11;
     else if (a == "q") col = 12;
     else if (a == "r") col = 13;
     else if (a == "s") col = 14;
     else if (a == "0") col = 15;
     else if (a == "1") col = 16;
     else if (a == "2") col = 17;
     else if (a == "3") col = 18;
     else if (a == "4") col = 19;
     else if (a == "5") col = 20;
     else if (a == "6") col = 21;
     else if (a == "7") col = 22;
     else if (a == "8") col = 23;
     else if (a == "9") col = 24;
     else if (a == "\"value=\"") col = 25;
     else if (a == "program") col = 26;
     else if (a == "var") col = 27;
     else if (a == "begin") col = 28;
     else if (a == "end.") col = 29;
     else if (a == "integer") col = 30;
     else if (a == "display") col = 31;
}

void getRow(string a, int& row)
{
     if (a == "P") row = 0;
     else if (a == "I") row = 1;
     else if (a == "A") row = 2;
     else if (a == "D") row = 3;
     else if (a == "U") row = 4;
     else if (a == "J") row = 5;
     else if (a == "T") row = 6;
     else if (a == "S") row = 7;
     else if (a == "C") row = 8;
     else if (a == "V") row = 9;
     else if (a == "W") row = 10;
     else if (a == "H") row = 11;
     else if (a == "X") row = 12;
     else if (a == "E") row = 13;
     else if (a == "E'") row = 14;
     else if (a == "Y") row = 15;
     else if (a == "Y'") row = 16;
     else if (a == "F") row = 17;
     else if (a == "N") row = 18;
     else if (a == "B") row = 19;
     else if (a == "Z") row = 20;
     else if (a == "Q") row = 21;
     else if (a == "R") row = 22;
}

void printStack(stack<string> myStack)
```

```
{
     stack<string> reverse;
     while (!myStack.empty())
     {
          reverse.push(myStack.top());
          myStack.pop();
     }
     cout << "Stack: ";
     while (!reverse.empty())
     {
          cout << reverse.top();
          reverse.pop();
     }
     cout << endl;
}

void parse(string filename)
{
     cout << "Testing program " << filename << " against grammar..." << endl << endl;
     fstream programInput;
     programInput.open(filename, ios::in);
     string token;
     programInput >> token;
     stack<string> stack1;
     stack1.push("$");
     stack1.push("P");
     printStack(stack1);
     bool done = false;
     while (!done)
     {
          string newToken;
          if (token == "\"value=\"" || token == "program" || token == "var" ||
               token == "begin" || token == "end." || token == "integer" ||
               token == "display")
          {

               newToken = token;
               pushAndPop(stack1, newToken, done);
          }
          else
          {
               for (int i = 0; i < token.size(); ++i)
               {
                    newToken = token[i];
                    pushAndPop(stack1, newToken, done);
               }
          }
          programInput >> token;
     }
}

void pushAndPop(stack<string>& stack1, string& newToken, bool& done)
{
     string table[23][32] = {
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15    16    17    18    19    20    21    22    23    24    25               26
27    28    29    30    31*/
          /*0*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "program I ; var D begin
S end.", "XX", "XX", "XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11      12
13     14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30    31*/
          /*1*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "R A", "R A",
"R A", "R A", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
```

```
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30    31*/
          /*2*/     {"XX", "&", "&", "&", "&", "&", "&", "&", "&", "XX", "&", "R A", "R A", "R A", "R
A", "Q A", "Q A", "Q A", "Q A", "Q A", "Q A", "Q A", "Q A", "Q A", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10     11
12          13          14       15    16    17    18    19    20    21    22    23    24    25    26
27    28    29    30    31*/
          /*3*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "U : T ;", "U
: T ;", "U : T ;", "U : T ;", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11      12
13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30     31*/
          /*4*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "I J", "I J",
"I J", "I J", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30     31*/
          /*5*/     {"XX", "XX", ", U", "XX", "XX", "XX", "&", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30      31*/
          /*6*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"integer", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10     11     12
13     14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30    31*/
          /*7*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "V C", "V C",
"V C", "V C", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "V C"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11   12   13
14    15    16    17    18    19    20    21    22    23    24    25    26    27    28   29    30
31*/
          /*8*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "S", "S", "S",
"S", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "&", "XX",
"S"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11   12   13
14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30
31*/
          /*9*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "X", "X", "X",
"X", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"W"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29
30         31*/
          /*10*/    {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "display ( H ) ;"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11   12   13
14    15    16    17    18    19    20    21    22    23    24        25           26    27    28
29    30    31*/
          /*11*/    {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "I", "I", "I",
"I", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "\"value=\" , I", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10     11
12          13          14       15    16    17    18    19    20    21    22    23    24    25    26
27    28    29    30    31*/
          /*12*/    {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "I = E ;", "I
= E ;", "I = E ;", "I = E ;", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "XX", "XX"},
```

```
          /*           0     1     2     3       4      5     6     7     8      9      10    11
12     13      14       15     16       17     18       19     20       21      22     23       24
25    26    27    28    29    30    31*/
          /*13*/     {"XX", "XX", "XX", "Y E'", "Y E'", "XX", "XX", "XX", "XX", "Y E'", "XX", "Y E'",
"Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y E'", "Y
E'", "XX", "XX", "XX", "XX", "XX", "XX"},
          /*           0     1     2     3         4       5     6     7     8     9    10    11
12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28
29    30    31*/
          /*14*/     {"XX", "&", "XX", "+ Y E'", "- Y E'", "XX", "XX", "XX", "XX", "XX", "&", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2     3       4      5     6     7     8      9      10    11
12     13      14      15      16      17      18      19      20      21      22      23      24
25    26    27    28    29    30    31*/
          /*15*/     {"XX", "XX", "XX", "F Y'", "F Y'", "XX", "XX", "XX", "XX", "F Y'", "XX", "F Y'",
"F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F Y'", "F
Y'", "XX", "XX", "XX", "XX", "XX", "XX", "XX"},
          /*           0     1     2     3     4     5     6      7         8        9    10    11
12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28
29    30    31*/
          /*16*/     {"XX", "&", "XX", "&", "&", "XX", "XX", "/ F Y'", "* F Y'", "XX", "&", "XX",
"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2     3   4     5     6     7     8      9      10    11   12   13
14   15   16   17   18   19   20   21   22   23   24   25    26    27    28    29    30    31*/
          /*17*/     {"XX", "XX", "XX", "N", "N", "XX", "XX", "XX", "XX", "( E )", "XX", "I", "I",
"I", "I", "N", "N", "N", "N", "N", "N", "N", "N", "N", "N", "XX", "XX", "XX", "XX", "XX", "XX", "XX"},
          /*           0     1     2     3         4       5     6     7     8     9    10    11
12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28
29    30    31*/
          /*18*/     {"XX", "XX", "XX", "Z Q B", "Z Q B", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "XX", "Q B", "Q B", "Q B", "Q B",  "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "XX", "XX",
"XX", "XX", "XX", "XX", "XX"},
          /*           0     1    2     3     4     5      6    7      8     9    10      11    12
13    14    15     16      17     18    19     20     21      22    23      24    25    26    27    28
29    30    31*/
          /*19*/     {"XX", "&", "XX", "&", "&", "XX", "XX", "&", "&", "XX", "&", "XX", "XX", "XX",
"XX", "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "Q B", "XX", "XX", "XX", "XX",
"XX", "XX", "XX"},
          /*           0     1     2    3    4     5     6     7     8     9    10    11    12    13
14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30
31*/
          /*20*/     {"XX", "XX", "XX", "+", "-", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "&", "&", "&", "&", "&", "&", "&", "&", "&", "&", "XX", "XX", "XX", "XX", "XX", "XX",
"XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10    11    12
13    14    15   16   17   18   19   20   21   22   23   24   25    26    27    28    29    30    31*/
          /*21*/     {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX", "XX", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "XX", "XX", "XX", "XX", "XX", "XX",
"XX"},
          /*           0     1     2     3     4     5     6     7     8     9    10   11   12   13
14   15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30
31*/
          /*22*/ {"XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "p", "q", "r",
"s", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX", "XX",
"XX"}
     };
     int rowNum = 0;
     int colNum = 0;
     bool match = false;
     while (!done && !match)
     {
          string pop = stack1.top();
          stack1.pop();
          while (pop == "")
```

```
        {
            pop = stack1.top();
            stack1.pop();
        }

        string terminal = newToken;
        cout << "Read: " << terminal << endl;
        getRow(pop, rowNum);
        getCol(terminal, colNum);
        string cfg = table[rowNum][colNum];
        while (!match && !done)
        {
            cout << "Pop: " << pop << endl;
            if (pop == terminal)
            {
                cout << "Match with input: " << terminal << endl;
                match = true;
                if (pop == "$")
                {
                    cout << "Program accepted!" << endl;
                    done = true;
                }
            }
            else if (pop == "XX")
            {
                cout << "Program rejected!" << endl;
                done = true;
            }
            else if (pop == "&")
            {
                pop = stack1.top();
                stack1.pop();
            }

            else
            {
                getRow(pop, rowNum);
                getCol(terminal, colNum);
                cfg = table[rowNum][colNum];
                addToStack(stack1, cfg);
                printStack(stack1);
                pop = stack1.top();
                stack1.pop();
            }
            while (pop == "")
            {
                pop = stack1.top();
                stack1.pop();
            }

        }
    }
}
```