```
//C++ Lexical Analyzer
//Sam Dressler
//CSCI 465
//9/26/2020
//This program takes in an input file and generates a list of lexeme-spelling pairs.
//The program takes the loads the file, generate symbols, and then classifies them into
// their appropriate lexemes.
//The output is generated into "output.txt"
#include "lexical_analyzer.h"
//Main Function to drive the modules.
int main(int argc, char * argv []){
  //Check to see if the user input a file
  if(argc != 2){
    cout << "Error: Enter the name of the pascal file to parse" << endl;
    exit(0);
  }
  //get the file name from user
  ifstream file(argv[1]);
  //Verify the file is opened.
  //If the file can not be opened, tell the user.
  if(!file.is_open()){
    cout << "Error file: "<<"\""<<argv[1]<<"\""<<" could not be opened"<<endl;
    exit(0);
  }
  //read from the file all of the characters and return to main
  char * file_as_chars = load_file(file);
  //Print raw input file
  cout << "--------------------------------"<<endl;
  cout << "Contents of: "<<"\""<<argv[1]<<"\""<<endl;
```

```
    cout << "--------------------------------"<<endl;
    cout << file_as_chars << endl;
    cout << "--------------------------------"<<endl;
    vector<symbol> symbol_table = vector<symbol>();
    vector<string> symbols = vector<string>();
    symbols = generate_symbols(file_as_chars);
    symbol_table = classify_symbols(symbols, symbol_table);
    FILE * output = NULL;
    output = fopen("output.txt","w");
      for(vector<symbol>::iterator it = symbol_table.begin(); it != symbol_table.end(); ++it)
          {
      symbol temp = *it;
      const char * token_t = temp.token_type.c_str();
      const char * token = temp.value.c_str();
      //cout << left << setw(12) << temp.token_type <<  " -->    " << temp.value << endl;
      fprintf(output,"%s %s\n",token_t, token);

    }
    cout << "Output file created" << endl;
    cout << "--------------------------------"<<endl;
    fclose(output);

    return 0;
}
```