

```

//Sam Dressler
//Header for Lex

//Includes
#include <iostream>
#include <fstream>
#include <string>
#include <string.h>
#include <stdio.h>
#include <vector>
#include <regex>
#include <iomanip>
#include "lex_token_types.h"

//Namepsace
using namespace std;

//Definitions
regex regex_identifier ("([_a-zA-Z])([a-zA-Z0-9]*){20}");
regex regex_number ("([0-9]*)");
string symbols = ".,:;()[{}]" ;
string operators = "+-*/";
int num_char=0;

struct symbol_type{
    //Lexeme
    string token_type;
    //Spelling
    string value;
};

```

```

typedef struct symbol_type symbol;

//Function definitions
char * load_file(ifstream& );
vector<string> generate_symbols(char *);
vector<symbol> classify_symbols(vector<string>, vector<symbol>);

/**
 * Function load_file
 * Description: Takes in a file and parses the characters into a dynamic
 * allocated array
 * Parameters: ifstream
 * Return char*
 */
char * load_file(ifstream &file){

    char temp_char;
    char* char_array = NULL;
    while(!file.eof())
    {
        file.get(temp_char);
        //process the characters
        if(!(temp_char == '\n'))
        {
            if(char_array == NULL)
            {
                char_array = new char[1];
                //cout << temp_char <<"+"<<num_char <<endl;
                char_array[num_char] = temp_char;
            }
        }
    }
}

```

```

        num_char++;
    }
    else
    {
        char * temp_char_array = char_array;
        char_array = new char[num_char+1];
        for(int i = 0; i < num_char; i++)
        {
            char_array[i] = temp_char_array[i];
        }
        //cout << temp_char <<" "<<num_char <<endl;
        char_array[num_char] = temp_char;
        num_char++;
        delete [] temp_char_array;
    }

}

//process \n
else
{
    if(char_array == NULL)
    {
        char_array = new char[1];
        char_array[0] = temp_char;
        num_char++;
    }
    else
    {
        char * temp_char_array = char_array;

```

```

        char_array = new char[num_char+1];
        for(int i = 0; i < num_char; i++)
        {
            char_array[i] = temp_char_array[i];
        }
        char_array[num_char] = temp_char;
        num_char++;
        delete [] temp_char_array;
    }
}

//cout << "Size of input file: "<< num_char << endl;
return char_array;
}

```

/**

* Function generate_symbols

* Description: take raw character output input and generate list of symbols

* Parameters: char *

* Return void

*/

```

vector <string> generate_symbols(char * raw_input){
    int width = 15;
    symbol curr_sym;
    vector<string> symbol_vec;

    //go through the raw input and split up by spaces or by a token
    char temp;
    char temp_look_ahead;
    string temp_word;

```

```

for (int i = 0; i < num_char; i++){
    temp = raw_input[i];
    if((i+1) < num_char){
        temp_look_ahead = raw_input[i+1];
        //cout << temp_look_ahead << endl;
    }

    if(temp == ' ' || temp == '\n'){
        if(temp_word != " " && temp_word != ""){
            //cout << temp_word << endl;
            symbol_vec.push_back(temp_word);
        }
        temp_word = "";
    }

    //check if temp is in the set of the special characters
    else if(symbols.find(temp) != std::string::npos){
        if(temp_word != " " && temp_word != ""){
            //cout << temp_word << endl;
            symbol_vec.push_back(temp_word);
        }
        if(temp == ':' && temp_look_ahead == '='){
            string temp_string ({temp ,temp_look_ahead});
            //cout << temp_string << endl;
            symbol_vec.push_back(temp_string);
            i += 1;
            temp_word = "";
            temp_string = "";
            continue;
        }
    }
}

```

```

    }
    else if(temp != ' ') {
        string temp_string ({temp});
        //cout << temp_string << endl;
        symbol_vec.push_back(temp_string);
    }
    temp_word = "";
}

else if(operators.find(temp) != std::string::npos){
    if(temp_word != " " && temp_word != ""){
        //cout << temp_word << endl;
        symbol_vec.push_back(temp_word);
    }
    if(temp != ' ') {
        string temp_string ({temp});
        //cout << temp_string << endl;
        symbol_vec.push_back(temp_string);
    }

    temp_word = "";
}

else if(temp == '='){
    if(temp_word != " " && temp_word != ""){
        //cout << temp_word << endl;
        symbol_vec.push_back(temp_word);
    }
    if(temp != ' '){
        string temp_string ({temp});
        //cout << temp_string << endl;
    }
}

```

```

        symbol_vec.push_back(temp_string);
    }
    temp_word = "";
}
else if(temp == '<'){
    if(temp_word != " " && temp_word != ""){
        //cout << temp_word << endl;
        symbol_vec.push_back(temp_word);
    }
    if(temp_look_ahead == '='){
        string temp_string ({temp ,temp_look_ahead});
        //cout << temp_string << endl;
        symbol_vec.push_back(temp_string);
        i+=1;
        temp_word = "";
        temp_string = "";
        continue;
    }
    else if(temp_look_ahead == '>'){
        string temp_string ({temp ,temp_look_ahead});
        //cout << temp_string << endl;
        symbol_vec.push_back(temp_string);
        i+=1;
        temp_word = "";
        temp_string = "";
        continue;
    }
    else if(temp != ' '){
        string temp_string ({temp});

```

```

        //cout << temp_string << endl;

        symbol_vec.push_back(temp_string);
    }

    temp_word = "";
}

else if(temp == '>'){
    if(temp_word != " " && temp_word != ""){
        //cout << temp_word << endl;

        symbol_vec.push_back(temp_word);
    }

    if(temp_look_ahead == '='){
        string temp_string ({temp ,temp_look_ahead});

        //cout << temp_string << endl;

        symbol_vec.push_back(temp_string);

        i+=1;

        temp_word = "";

        temp_string = "";

        continue;
    }

    else if(temp != ' '){
        string temp_string ({temp});

        //cout << temp_string << endl;

        symbol_vec.push_back(temp_string);
    }

    temp_word = "";
}

else{
    temp_word += temp;
}

```



```

    }
    return symbol_vec;
}

```

```

string trim(string string)
{
    size_t pos = string.find_first_not_of(" ");
    string.erase(0, pos);

    pos = string.find_last_not_of(" ");
    if (string::npos != pos)
        string.erase(pos+1);
    return string;
}

```

```

/**
 * Function: classify_symbols - determine what tokentype
 *
 *                               each of the symbols are and create an
 *                               entry in the symbol table with
 * tokentype and then
 *
 *                               include what the value of the token is.
 */

```

```

vector<symbol> classify_symbols(vector<string> symbol_array, vector<symbol> symbol_table)
{
    int i = 0;
    int width = 12;
    int size = symbol_array.size();
    string value;

```

```

for(vector<string>::iterator it = symbol_array.begin(); it != symbol_array.end()-1; ++it)
{
    value = *it;
    symbol s;

    s.value = value;
    //cout <<"-----> "<< s.value << endl;

    if(symbol_array[i].compare("") == 0){
        continue;
    }
    else if(symbol_array[i].compare(" ") == 0){
        continue;
    }
    else if (symbol_array[i].compare("and") == 0){
        s.token_type = "and_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("array") == 0){
        s.token_type = "array_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("begin") == 0){
        s.token_type = "begin_sym";

```

```

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("char") == 0){

        s.token_type = "char_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("chr") == 0){

        s.token_type = "chr";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("do") == 0){

        s.token_type = "do_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("else") == 0){

        s.token_type = "else_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("end") == 0){

        s.token_type = "end_sym";

```

```

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("if") == 0){

        s.token_type = "if_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("int") == 0){

        s.token_type = "integer_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("integer") == 0){

        s.token_type = "integer_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("mod") == 0){

        s.token_type = "mod_sym";

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("not") == 0){

        s.token_type = "not_sym";

```

```

        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    //what is this
    else if(symbol_array[i].compare("of") == 0){
        s.token_type = "of_sym";
        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("or") == 0){
        s.token_type = "or_sym";
        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    //what is this
    else if(symbol_array[i].compare("ord") == 0){
        s.token_type = "ord_sym";
        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("procedure") == 0){
        s.token_type = "procedure_sym";
        symbol_table.push_back(s);

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

```

```

else if(symbol_array[i].compare("function") == 0){
    s.token_type = "function_sym";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("program") == 0){
    s.token_type = "program_sym";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("read") == 0){
    s.token_type = "read";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("readln") == 0){
    s.token_type = "readln_sym";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("then") == 0){
    s.token_type = "then_sym";
    symbol_table.push_back(s);

```

```

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("var") == 0){
        s.token_type = "var_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("while") == 0){
        s.token_type = "while_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("write") == 0){
        s.token_type = "write_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("writeln") == 0){
        s.token_type = "writeln_sym";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("+") == 0){
        s.token_type = "plus";

```

```

        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("-") == 0){
        s.token_type = "minus";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("*") == 0){
        s.token_type = "times";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("/") == 0){
        s.token_type = "div";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
    }

    else if(symbol_array[i].compare("<") == 0){
        s.token_type = "less";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;;
    }

```



```
        else if(symbol_array[i].compare("<=") == 0){
            s.token_type = "lessequal";
            symbol_table.push_back(s);
            //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
        }
```

```
        else if(symbol_array[i].compare("<>") == 0){
            s.token_type = "notequal";
            symbol_table.push_back(s);
            //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
        }
```

```
        else if(symbol_array[i].compare(">") == 0){
            s.token_type = "greater";
            symbol_table.push_back(s);
            //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
        }
```

```
        else if(symbol_array[i].compare(">=") == 0){
            s.token_type = "greaterequal";
            symbol_table.push_back(s);
            //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
        }
```

```
        else if(symbol_array[i].compare("=") == 0){
            s.token_type = "equals";
```

```

        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " -->  " << s.value <<
endl;
    }

    else if(symbol_array[i].compare(":=") == 0){
        s.token_type = "assign";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " -->  " << s.value <<
endl;
    }

    else if(symbol_array[i].compare(":") == 0){
        s.token_type = "colon";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " -->  " << s.value <<
endl;
    }

    else if(symbol_array[i].compare(";") == 0){
        s.token_type = "semicolon";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " -->  " << s.value <<
endl;
    }

    else if(symbol_array[i].compare(",") == 0){
        s.token_type = "comma";
        symbol_table.push_back(s);

```

```

        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare(".") == 0){
        s.token_type = "period";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("(") == 0){
        s.token_type = "lparen";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare(")") == 0){
        s.token_type = "rparen";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

    else if(symbol_array[i].compare("[") == 0){
        s.token_type = "lbrack";
        symbol_table.push_back(s);
        //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

    }

```

```

else if(symbol_array[i].compare("]") == 0){
    s.token_type = "rbrack";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("{") == 0){
    s.token_type = "lbrace";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

else if(symbol_array[i].compare("}") == 0){
    s.token_type = "rbrace";
    symbol_table.push_back(s);
    //cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

//Handle the identifier, number, quotestring, litchar, eofsym, and
//illegal tokens
//check if the string is a valid identifier
else if(regex_match(value, regex("[a-zA-Z]{1}"))){
    s.token_type = "litchar";
    symbol_table.push_back(s);
    ////cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;

}

```

```

else if(regex_match(value, regex_identifier)){
    s.token_type = "identifier";
    symbol_table.push_back(s);
    ////cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;;
}
else if(regex_match(value, regex_number)){
    s.token_type = "number";
    symbol_table.push_back(s);
    ////cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
}
else if(regex_match(value, regex("\"([a-zA-Z0-9+)]\"") ))){
    s.token_type = "quotestring";
    symbol_table.push_back(s);
    ////cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
}
else if(i == size-1){
    s.token_type = "eofsym";
    symbol_table.push_back(s);
    ////cout << left << setw(width) << s.token_type << " --> " << s.value <<
endl;
}
else{
    if(regex_match(value, regex(".*"))){

        cout << value << endl;
        s.token_type = "illegal";
        symbol_table.push_back(s);

```

```
        continue;
    }
}
i++;
}

return symbol_table;
}
```