

Computer Science 323
Fall 2018

Final Project

Group Members:

Theresa Tanubrata

Brian Trinh

Dominick Weaver

Method used: Top-Down Parser (Table I)

Language Use: C++

Original Program

<prog>	→	program <id> ; var <dec-list> begin <stat-list> end
<id>	→	<letter> { <letter> <digit> }
<dec-list>	→	<dec> : <type> ;
<dec>	→	<id> , <dec> <id>
<type>	→	integer
<stat-list>	→	<stat> <stat> <stat-list>
<stat>	→	<write> <assign>
<write>	→	show (<id>) ;
<assign>	→	<id> = <expr> ;
<expr>	→	<expr> + <term> <expr> - <term> <term>
<term>	→	<term> * <factor> <term> / <factor> <factor>
<factor>	→	<id> <number> (<expr>)
<number>	→	<sign> <digit> { <digit> }
<sign>	→	+ - λ
<digit>	→	0 1 2 ... 9
<letter>	→	a b c d e

Grammar in BNF form

<prog>	→	program <id> ; var <dec-list> begin <stat-list> end
<id>	→	<letter> <id'>
<id'>	→	<letter> <id'>
<id'>	→	<digit> <id'>
<id'>	→	λ
<dec-list>	→	<dec> : <type> ;
<dec>	→	<id> <dec'>
<dec'>	→	, <id> <dec'>
<dec'>	→	λ
<type>	→	integer
<stat-list>	→	<stat> <stat-list'>
<stat-list'>	→	<stat> <stat-list'>
<stat-list'>	→	λ
<stat>	→	<write>
<stat>	→	<assign>
<write>	→	show (<id>) ;
<assign>	→	<id> = <expr> ;
<expr>	→	<term> <expr'>
<expr'>	→	+<term> <expr'>
<expr'>	→	-<term> <expr'>
<expr'>	→	λ
<term>	→	<factor> <term'>
<term'>	→	* <factor> <term'>
<term'>	→	/ <factor><term'>
<term'>	→	λ
<factor>	→	<id>
<factor>	→	<number>
<factor>	→	(<expr>)
<number>	→	<sign> <digit> <number'>
<number'>	→	<digit> <number'>
<number'>	→	λ
<sign>	→	+
<sign>	→	-
<sign>	→	λ
<digit>	→	0
<digit>	→	1
<digit>	→	2
<digit>	→	3
<digit>	→	4
<digit>	→	5
<digit>	→	6
<digit>	→	7
<digit>	→	8

<digit>	→	9
<letter>	→	a
<letter>	→	b
<letter>	→	c
<letter>	→	d
<letter>	→	e

Members of FIRST and FOLLOW

Original Non-Terminals	New Name	FIRST	FOLLOW
<prog>	P	program	\$
<id>	I	a b c d e	; , = : * / + -)
<id2>	K	a b c d e 0 1 2 3 4 5 6 7 8 9 λ	; , = : * / + -)
<dec-list>	D	a b c d e	begin
<dec>	B	a b c d e	:
<dec2>	M	, λ	:
<type>	C	integer	;
<stat-list>	G	show a b c d e	end
<stat>	S	show a b c d e	show a b c d e end
<write>	W	show	show a b c d e end
<assign>	A	a b c d e	show a b c d e end
<expr>	E	a b c d e + - 0 1 2 3 4 5 6 7 8 9 () ;
<expr'>	Q	+ - λ) ;
<term>	T	a b c d e + - 0 1 2 3 4 5 6 7 8 9 (+ -) ;
<term'>	R	* / λ	+ -) ;
<factor>	F	a b c d e + - 0 1 2 3 4 5 6 7 8 9 (* / + -) ;
<number>	N	+ - 0 1 2 3 4 5 6 7 8 9	* / + -) ;
<number2>	O	0 1 2 3 4 5 6 7 8 9 λ	* / + -) ;
<sign>	H	+ - λ	0 1 2 3 4 5 6 7 8 9
<digit>	J	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 * / + -) ;
<letter>	L	a b c d e	a b c d e 0 1 2 3 4 5 6 7 8 9 ; , = : * / + -)
<stat-list2>	U	show a b c d e λ	end

[illegible]

```

//
// Header file, Vector_String containing functions for Part I, II, and, III
//

#ifndef VECTOR_STRING_INCLUDE
#define VECTOR_STRING_INCLUDE

#include <iostream>
#include <fstream>           //file reading
#include <string>
#include <vector>
#include <sstream>

#include <stdlib.h>

//define PAUSE cout << "\n\n"; system("pause");

using namespace std;
//PART I FUNCTIONS
void readfile(string &); // read a file
void remove_white_space(string &); // remove whitespace at the beginning of strings
void remove_single_comments(string &); // remove single line comments
void remove_multiple_comments(fstream&, string&); // remove multiple line comments
void proper_spacing(string&); // add proper spacing to the line
void program_vector(vector<string>&); // populate a vector with the tokens found in a program file
void walk_stack(vector<string>&); //used to debug vectors

//PART II FUNCTIONS
bool check_file_for_errors(vector<string>&); //Check file for any automatic errors
bool check_spelling(string&); //Check spelling for integer and word

//PART III FUNCTION
void create_cpp_file(vector<string>&); //Translate finalp2.txt file into c++ program

// & in parameter to pass by reference and manipulate string::file_contents data
void readfile(string &file_contents) { // a function to read a file into a string
    fstream ifile;

    ifile.open("/Users/Theresa/CLionProjects/untitled13/finalp1.txt");

    if (!ifile.is_open()) {
        std::cout << "finalp1.txt NOT FOUND...\n\n";
        //PAUSE
    }
    //else cout << "finalp1.txt FOUND!\n\n";

    string str_line;

    while (!ifile.eof()) {
        getline(ifile, str_line);

        remove_white_space(str_line);           // remove white space

        remove_multiple_comments(ifile, str_line); // REMOVE MULTIPLE LINE COMMENTS
    }
}

```

```

remove_single_comments(str_line);                                // remove single line comments

proper_spacing(str_line);                                       // ADD PROPER SPACING

if (str_line.length() > 0 &&
    // if length is greater than 0
    str_line.find_first_not_of(' ') != std::string::npos &&
    str_line.find_first_not_of("\t") != std::string::npos) {    // if the line doesnt consist of space
                                                                // if the line doesnt consist of tabs

    while (str_line.back() == ' ') {
        str_line.pop_back();
    }

    if (!str_line.find("end")) {
        file_contents += str_line;
    } else file_contents += str_line + "\n";
}
}

//cout << "\tSTRING CONTENTS FOR FILE AFTER\n\t\tt-WHITE SPACE REMOVAL\n\t\tt-SINGLE LINE
COMMENT REMOVAL\n\t\tt-MULTILINE COMMENT REMOVAL\n\n" << file_contents; //
string::file_contents population test
ifile.close();

ofstream ofile;

ofile.open("finalp2.txt");
ofile << file_contents;
ofile.close();

} // end void readfile(string &)

void remove_white_space(string &str_line) {
    // REMOVING LEADING WHITE SPACE

    // this function checks to see if the line is all white_space, checking for space and tab
    // by default, std::string.find_first_not_of(<char>) will return std::string::npos (the maximum amount of
    characters in a string)
    // if std::string::npos (the maximum amount of characters in a string) is not returned, then there was a character
    other than space ' ' or tab '\t'
    if (str_line.find_first_not_of(' ') != std::string::npos &&
        str_line.find_first_not_of("\t") != std::string::npos) {

        //cout << "BEFORE " << str_line.length() << " : " << str_line << endl; // DEBUG

        for (int element = 0; str_line[element];) {

            if (isspace(str_line[element])) {
                while (isspace(str_line[element])) {
                    //cout << "\tSPACE FOUND\n"; // DEBUG
                    element++;
                }
                continue;
            }

            if (!isspace(str_line[element])) {

```



```

        if (element != 0) {
            //cout << "\tERASING " << element << "ELEMENTS!" << endl; // DEBUG
            str_line.erase(str_line.begin(), str_line.begin() + element); // erase from beginning to a position
        }
        break;
    }

}

//cout << "AFTER " << str_line.length() << " : " << str_line << "\n\n";
} // end leading white space if statement check

//          DELETE EXTRA WHITE SPACES IN LINES

for (int element = 0; element < str_line.length(); element++) { // walk the string
    if (isspace(str_line[element])) {
        while (isspace(str_line[element + 1])) {
            str_line.erase(str_line.begin() + element);
            //cout << "DELETING EXTRA WS\n"; // DEBUG
        }
    }
}

}

void remove_single_comments(string &str_line) {
    //          REMOVING SINGLE LINE ("//") COMMENTS

    size_t found_at = 0;
    int elements = str_line.length(); // length of the string

    found_at = str_line.find("//"); // by default, the std::string.find(<string>) will return std::string::npos (the
    maximum amount of characters in a string)

    if (found_at != std::string::npos) {

        // cout << "<// FOUND AT> : " << found_at << "\n"; // std::string.find() functionality test // DEBUG

        //the following occurs while the length of the string is not equal to the 0th position that the "//" string was
        found
        //it will remove the comment
        while (str_line.length() != found_at) {
            str_line.pop_back(); // pop until there are no more white spaces
        }
    }
}

void remove_multiple_comments(fstream &ifile, string &str_line) {

    bool multi = 0;

    size_t start_found_at = 0;
    size_t end_found_at = 0;

    //          REMOVE MULTIPLE LINE COMMENTS

    start_found_at = str_line.find("/*"); // by default, the std::string.find(<string>) will return std::string::npos (the
    maximum amount of characters in a string)

```

```

if (start_found_at != std::string::npos) {

    //cout << "Start Multi Comment Found\n"; // DEBUG
    //cout << str_line << endl; // DEBUG

    multi = 1; // start multiple line comment symbol found

    end_found_at = str_line.find("*/"); // see if "*/" is in this line
    if (end_found_at != std::string::npos) {
        str_line.erase(str_line.begin() + start_found_at, str_line.begin() + end_found_at + 1); // Erase everything
        from the beginning of the line up to and including the / in */

        multi = 0; // multiple line comment end symbol found. multiple line comment resolved
    }

    else { // if "*/" is not in the line

        while (multi) { // while multiple line comment end symbol not resolved

            getline(ifile, str_line); // grab the next line
            //cout << str_line << endl; // DEBUG
            end_found_at = str_line.find("*/");

            if (end_found_at != std::string::npos) {
                //cout << "End Multi Comment Found\n"; // DEBUG
                str_line.erase(str_line.begin(), str_line.begin() + end_found_at + 2);
                multi = 0;
            }
        }
    }
} // END REMOVE MULTIPLE LINE COMMENTS
}

void proper_spacing(string &str_line) {
    // ADD PROPER SPACING

    for (int element = 0; element < str_line.length(); element++) { // walk the string

        // symbol exception ex +2
        // break if the next character is a number
        if (element != 0 &&
            (str_line[element] == '+' ||
             str_line[element] == '-') &&
            isdigit(str_line[element + 1])) {

            break;
        }

        // if there is a punctuation before another token without spaces in between (example: ",TOKEN")
        // add a space where the token begins; between the punctuation and the next token (" ,TOKEN")
        if (element != 0 && // not the beginning
            isalnum(str_line[element]) && // current is number or letter
            ispunct(str_line[element - 1])) { // previous is a punctuation

            str_line.insert(str_line.begin() + element, ' ');
            //element += 1; // increment by 1 because the string increases by one and following chars from this
            position have their address incremented by one
        }
    }
}

```

```

        // if there is a punctuation and it is not the first element of the string
        // add a space before it if there doesnt already exist a space character
        if (element != 0 &&                                     // not the beginning
            ispunct(str_line[element]) &&                       // current is punctuation
            str_line[element - 1] != ' ') {                     // previous is not a space

            str_line.insert(str_line.begin() + element, ' ');
            //element += 1; // increment by 1 because the string increases by one and following chars from this
            position have their address incremented by one
        }
    }
}

```

```

void program_vector(vector<string>& vector_given_tokens) {

```

```

    // CREATING TOKENS FROM THE FILE AND PLACING THEM IN A VECTOR

```

```

    ifstream ifile2;
    ifile2.open("finalp2.txt");

```

```

    if (!ifile2.is_open()) {
        cout << "finalp2.txt not found...\n";
        //PAUSE
        //    exit;
    }

```

```

    //cout << "\nfinalp2.txt FOUND!\n";
    string str_line;
    string word;

```

```

    while (!ifile2.eof()) {
        getline(ifile2, str_line);
        for (int x = 0; x < str_line.length(); x++) {
            if (str_line[x] == ' ' || x + 1 == str_line.length()) {

                if (x + 1 == str_line.length()) {
                    word += str_line[x];
                }

                while (word.back() == ' ') {
                    word.pop_back();
                }

                vector_given_tokens.push_back(word);
                word.clear();

                continue;
            }

            word += str_line[x];
        }
    }

```

```

    //    VECTOR POPULATION CHECK

```

```

    /*for (int x = 0; x < vector_words.size(); x++) {
        cout << x << ".\t" << vector_words[x] << "\n";
    }*/

```

```
}
```

```
void walk_stack(vector<string>& state_stack){  
    vector<string> temp = state_stack;  
    cout << "Current Stack: ";  
    for(auto it = temp.begin(); it < temp.end(); it++){  
        cout << *it << " \t";  
    }  
    cout << endl;  
}
```

```
bool check_file_for_errors(vector<string>& vector_given_tokens){  
    //counters for the parentheses  
    int p_open = 0, p_close = 0;  
    //temporary vector string to keep track of declared variables  
    vector<string> temp;  
  
    //checks if 'program' is in the file  
    if(vector_given_tokens[0] != "program"){  
        cout << endl;  
        cout << "program \tis expected\n";  
        return true;  
    }  
  
    //checks if 'end' is in the file  
    if(vector_given_tokens[vector_given_tokens.size()-1] != "end"){  
        cout << endl;  
        cout << "end \tis expected\n";  
        return true;  
    }  
  
    //loops through the file to find any undefined variables, or illegal expressions  
    //also counts how many open and closed parantheses there are  
    for(int i = 0; i < vector_given_tokens.size(); i++){  
        if(vector_given_tokens[i] == "("){  
  
            //counts left parentheses  
            p_open++;  
        }else if(vector_given_tokens[i] == ")"){  
  
            //counts right parentheses  
            p_close++;  
        }  
  
        if(vector_given_tokens[i] == "var"){  
  
            //looks for variables and pushes them on a temporary vector  
            //increments to the beginning of the declaration list  
            i++;  
            bool dec_list_done = false;  
  
            //keeps looping through the declaration list until it reaches the end (: or integer)  
            while(!dec_list_done){
```

```

//if reading : or integer then the declaration list of variables is done
//put : and integer and ; in case one of them is missing from the file
if(vector_given_tokens[i] == ":" || vector_given_tokens[i] == "integer" || vector_given_tokens[i] ==
";"){
    dec_list_done = true;
}

//pushes back only the variables and no punctuation marks
if(vector_given_tokens[i] != "," && vector_given_tokens[i] != ":") {
    temp.push_back(vector_given_tokens[i]);
}
i++;
}
}

```

//looping thorough part that uses the declared variables: assigning values, showing values, and changing values

```

//Mainly looks for any illegal expressions or undefined variables being used
if(vector_given_tokens[i] == "begin" && !temp.empty()){

```

```

    int k = i+1;
    //current variable being looked at
    string var_name;
    //string used to check if the expression is legal
    string check_expression;

```

```

while(vector_given_tokens[k] != "end"){
    var_name = vector_given_tokens[k];
    auto it = var_name.begin();
    if( (*it >='a' && *it <='e') || (*it >='0' && *it <='9') ){
        //specifically checks the variables
        if(*it >='a' && *it <='e'){
            bool defined = false;
            for(int l = 0; l < temp.size(); l++){
                if(temp[l] == var_name){
                    defined = true;
                }
            }
            if(!defined){
                cout << endl;
                cout << var_name << "\tUndefined Expression!\n";
                return true;
            }
        }
    }
}

```

```

//checks whether expression is valid
//this iterates to the next token
check_expression = vector_given_tokens[k+1];
auto it2 = check_expression.begin();
//checks if the next token after a variable or a number is a variable, number or '('
if( (*it2 >='a' && *it2 <='e') || *it2 == '(' || (*it2 >='0' && *it2 <='9')){
    cout << endl;
    cout << "Illegal Expression\n";
    return true;
}

```

//this checks other illegal expressions e.g. '2 * + e338', 'e338 *)'

```

    }else if(*it == '+' || *it == '*' || *it == '()){
        check_expression = vector_given_tokens[k+1];
        auto it3 = check_expression.begin();

        if(*it3 == '+' || *it3 == '*' || *it3 == '()){
            cout << endl;
            cout << "Illegal Expression\n";
            return true;
        }
    }
    k++;
}

}

}

//checking if the left parentheses or right parentheses is more than the other
if(p_open < p_close){
    cout << endl;
    cout << "\t Left Parentheses is missing.\n";
    return true;
}else if(p_open > p_close){
    cout << endl;
    cout << "\t Right Parentheses is missing.\n";
    return true;
}

//returns false if there are no automatic errors in the file
return false;
}

//Only checks the spelling of the words integer and show
bool check_spelling(string& word){
    //checks to see if any of the letters in the referenced word contains any of the letters from the words "integer"
    or "show"

    for(auto it = word.begin(); it != word.end(); ++it){
        if(*it == 'i' || *it == 'n' || *it == 't' || *it == 'e' || *it == 'g' || *it == 'r'){
            cout << endl;
            cout << "integer is misspelled!\n";
            return true;
        }else if(*it == 's' || *it == 'h' || *it == 'o' || *it == 'w'){
            cout << endl;
            cout << "show is misspelled!\n";
            return true;
        }
    }
    return false;
}

```

```
//PART III FUNCTION
```

```
//CREATING THE FILEP3.CPP
```

```
void create_cpp_file(vector<string>& vector_given_tokens){
    // Create .cpp file for output
    fstream ifile3;
    ifile3.open("filep3.cpp", ios::out);

    // bool flag for tabs within "int main()"
    bool tabFlag = false;

    cout << "\nCreating new filep3.cpp\n\n";

    // Begin iterating through vector
    for (int y = 0; y < vector_given_tokens.size(); y++) {

        // Rule for keyword "program"
        if (vector_given_tokens[y] == "program") {
            ifile3 << "#include <iostream>" << "\n"
                << "using namespace std;" << "\n";

            while (vector_given_tokens[y] != ";") // Increment vector position while ';' is not found
                y++;

        }

        // Rule for keyword "var"
        else if (vector_given_tokens[y] == "var") {
            y += 1; // Skip "var" token

            do {
                if (vector_given_tokens[y] == "integer") // Shorten "integer" into "int".
                    ifile3 << "int" << ' ';
                else
                    ifile3 << vector_given_tokens[y] << ' ';

                y += 1; // Increment vector position

            } while (vector_given_tokens[y] != ";"); // Exit loop once a ';' is found

            ifile3 << ";\n"; // Finish "var" rule by including this string

        }

        // Rule for keyword "begin"
        else if (vector_given_tokens[y] == "begin") {
            ifile3 << "int main()" << "\n" // Ouput main function to .cpp
                << '{' << "\n\t";
            tabFlag = true; // Begin tabbing each line

        }

        // Rule for "show"
        else if (vector_given_tokens[y] == "show") {
```

```

    ifile3 << "cout << ";

    y += 2;          // Skip ahead two vector positions to bypass "show" and "("

    ifile3 << vector_given_tokens[y];

    y += 1;          // Skip ahead one vector position to bypass ")"

    ifile3 << ' ';
}

// Rule for ';'
else if (vector_given_tokens[y] == ";" || vector_given_tokens[y] == ";") {    // Question: Why do I need
to check for "; "?
    ifile3 << ";\n";

    if (tabFlag == true && vector_given_tokens[y + 1] != "end")    // tabFlag is true only after .cpp file
has entered "int main() function.
        ifile3 << "\t";    // Second condition used so the last
        "}" is NOT tabbed.
    }

// Rule for "end"
else if (vector_given_tokens[y] == "end") {
    tabFlag = false;    // Reset tabFlag to false
    ifile3 << ";\n";    // Finish .cpp file
}

else
    ifile3 << vector_given_tokens[y] << ' ';
}

cout << "File filep3.cpp has been created.\n";

ifile3.close();
}

#endif

```



```

//
// Main CPP File, Runs PART II Functions
//

#include <iostream>
#include <string>
#include <vector>
#include <iostream>
#include <fstream>
//LIBRARY THAT IGNORES/TAKES OUT WHITE SPACES
#include <sstream>
#include <string>
#include <stdlib.h>
#include "VectorString.h"

using namespace std;

//first and follow table
string table[22][19] = {

    //state program          integer show( ; var begin end a-e 0-9 + - ( *
    / , : ); = ; invalid
/*P=0*/ {"program I; var D begin G end", "null", "null", "null", "null", "null", "null", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null", "null", "null"},
/*I=1*/ {"null", "null", "null", "null", "null", "null", "LK", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*K=2*/ {"null", "null", "null", "null", "null", "null", "LK", "JK", "lambda", "lambda",
"null", "lambda", "lambda", "lambda", "lambda", "lambda", "lambda", "lambda", "null"},
/*D=3*/ {"null", "null", "null", "null", "null", "null", "B:C;", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*B=4*/ {"null", "null", "null", "null", "null", "null", "IM", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null"},
/*M=5*/ {"null", "null", "null", "null", "null", "null", "null", "null", "null", "null", "null",
"null", "null", "IM", "lambda", "null", "null", "null", "null"},
/*C=6*/ {"null", "integer", "null", "null", "null", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*G=7*/ {"null", "null", "SU", "null", "null", "null", "SU", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*S=8*/ {"null", "null", "W", "null", "null", "null", "A", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null"},
/*W=9*/ {"null", "null", "show(I);", "null", "null", "null", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*A=10*/ {"null", "null", "null", "null", "null", "null", "I=E;", "null", "null", "null",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*E=11*/ {"null", "null", "null", "null", "null", "null", "TQ", "TQ", "TQ", "TQ",
"TQ", "null", "null", "null", "null", "null", "null", "null"},
/*Q=12*/ {"null", "null", "null", "null", "null", "null", "null", "null", "+TQ", "-TQ",
"null", "null", "null", "null", "null", "lambda", "null", "lambda", "null"},
/*T=13*/ {"null", "null", "null", "null", "null", "null", "FR", "FR", "FR", "FR", "FR",
"null", "null", "null", "null", "null", "null", "null"},
/*R=14*/ {"null", "null", "null", "null", "null", "null", "null", "null", "lambda", "lambda",
"null", "FR", "FR", "null", "null", "lambda", "null", "lambda", "null"},
/*F=15*/ {"null", "null", "null", "null", "null", "null", "null", "null", "I", "N", "N", "N", "(E)",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*N=16*/ {"null", "null", "null", "null", "null", "null", "null", "null", "JO", "HJO", "HJO",
"null", "null", "null", "null", "null", "null", "null", "null"},
/*O=17*/ {"null", "null", "null", "null", "null", "null", "null", "null", "JO", "lambda", "lambda",
"null", "lambda", "lambda", "null", "null", "lambda", "null", "lambda", "null"}

```

```

/*H=18*/{"null" , "null", "null" , "null" , "null", "null", "null", "lambda", "+", "-",
"null", "null" , "null" , "null" , "null" , "null" , "null" , "null", "null"},
/*J=19*/{"null" , "null", "null" , "null" , "null", "null", "0-9" , "null" , "null" , "null",
"null" , "null" , "null" , "null" , "null" , "null" , "null", "null"},
/*L=20*/{"null" , "null", "null" , "null" , "null", "null", "a-e" , "null", "null" , "null" , "null",
"null" , "null" , "null" , "null" , "null" , "null" , "null", "null"},
/*U=21*/{"null" , "null", "SU" , "null" , "null", "lambda", "SU" , "null", "null" , "null" ,
"null", "null" , "null" , "null" , "null" , "null" , "null", "null"}

};
//vector_given_tokens is where you read the file,
//state_stack is where you push and pop the states to accept the file
vector<string> state_stack;
//Keep track of elements in the state_stack(the popped element)
string state;
int row, column;

//Assigns the rows and does state = [row][column]
//finds match for current state
//puts out error messages if null
vector<string> update_state(vector<string>, string&);

//Used to iterate individual letters in string e.g ab1, ce334, etc
//assigns the columns
void check_var(string);

//Used to iterate through reserved_words and other terminals
//assigns the columns
void check_word(string);

int main(){

//PART I
if (remove("finalp2.txt") == 0) {
    cout << "\n\n\tPrevious finalp2.txt file found and deleted\n\n";
    //PAUSE
    //    system("cls");
}

string file_contents;

readfile(file_contents);

vector<string> vector_given_tokens;

// CREATING TOKENS AND PLACING THEM IN A VECTOR

program_vector(vector_given_tokens);

//<DEBUGGING STACK>
//walk_stack(vector_given_tokens);

cout << "Starting Part II.\n";

```

```

//this string will be used for the words read in the file
string iter;
//pushing first element onto the stack
state_stack.push_back("P");

//if true then there are automatic errors in the file e.g. program or end missing, etc.
bool errors = check_file_for_errors(vector_given_tokens);
if(errors){
    cout << "Rejected.\n";
    return 0;
}

cout << "<READING PROGRAM>\n";
for(int i=0; i<vector_given_tokens.size(); i++){
    //pop top element in state_stack
    state = state_stack.back();
    state_stack.pop_back();
    iter = vector_given_tokens[i];

    check_word(iter);

    if(state == "null") break;

}

if(state != "null") {
    cout << "No error.\nAccepted!\n";
    create_cpp_file(vector_given_tokens);

} else cout << "Rejected. Cannot move on to Part III\n";

return 0;
}

```

//PART II FUNCTIONS

```

void check_var(string iter){
    auto it = iter.begin();
    string letter;
    //assigning and matching for each letter in the word to match
    do{
        //a-e is column = 6 and 0-9 is column 7
        switch(*it){
            case 'a':
            case 'b':
            case 'c':
            case 'd':
            case 'e': column = 6; break;

```

```

        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9': column = 7; break;

    }

    //making char a string
    letter = *it;
    state_stack = update_state(state_stack, letter);
    it++;

} while(it != iter.end() && state != "null");
}

```

```

void check_word(string iter){

    //assigning columns
    if(iter == "program"){
        column = 0;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "integer"){
        column = 1;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "show"){
        column = 2;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "var"){
        column = 3;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "begin"){
        column = 4;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "end"){
        column = 5;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "+"){
        column = 8;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "-"){
        column = 9;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "("){
        column = 10;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "*"){
        column = 11;
        state_stack = update_state(state_stack, iter);
    } else if(iter == "/"){
        column = 12;
        state_stack = update_state(state_stack, iter);
    } else if(iter == ","){

```

```

        column = 13;
        state_stack = update_state(state_stack, iter);
    } else if (iter == ":") {
        column = 14;
        state_stack = update_state(state_stack, iter);
    } else if (iter == ")") {
        column = 15;
        state_stack = update_state(state_stack, iter);
    } else if (iter == "=") {
        column = 16;
        state_stack = update_state(state_stack, iter);
    } else if (iter == ";") {
        column = 17;
        state_stack = update_state(state_stack, iter);
    } else { //this checks the individual letters and ints in the variable e.g. ab1, e33a, b16
        check_var(iter);
    }
}
}

```

```

vector<string> update_state(vector<string> state_stack, string& iter) {
    bool word_match = false;
    while (!word_match) {
        //assigning the row and reading the table with row and column
        if (state == "A") {
            row = 10;
            state = table[row][column];
        } else if (state == "B") {
            row = 4;
            state = table[row][column];
        } else if (state == "C") {
            row = 6;
            state = table[row][column];
        } else if (state == "D") {
            row = 3;
            state = table[row][column];
        } else if (state == "E") {
            row = 11;
            state = table[row][column];
        } else if (state == "F") {
            row = 15;
            state = table[row][column];
        } else if (state == "G") {
            row = 7;
            state = table[row][column];
        } else if (state == "H") {
            row = 18;
            state = table[row][column];
        } else if (state == "I") {
            row = 1;
            state = table[row][column];
        } else if (state == "J") {
            row = 19;
            state = table[row][column];
        } else if (state == "K") {
            row = 2;

```

```

        state = table[row][column];
    } else if (state == "L") {
        row = 20;
        state = table[row][column];
    } else if (state == "M") {
        row = 5;
        state = table[row][column];
    } else if (state == "N") {
        row = 16;
        state = table[row][column];
    } else if (state == "O") {
        row = 17;
        state = table[row][column];
    } else if (state == "P") {
        row = 0;
        state = table[row][column];
    } else if (state == "Q") {
        row = 12;
        state = table[row][column];
    } else if (state == "R") {
        row = 14;
        state = table[row][column];
    } else if (state == "S") {
        row = 8;
        state = table[row][column];
    } else if (state == "T") {
        row = 13;
        state = table[row][column];
    } else if (state == "U") {
        row = 21;
        state = table[row][column];
    } else if (state == "W") {
        row = 9;
        state = table[row][column];
    }
}

if (state == iter) { //SUCCESSFUL MATCH
    cout << "Successful match: " << iter << endl;
    word_match = true;
} else if (state == "program I; var D begin G end") { //pushing back elements from the table into the
state_stack

    state_stack.push_back("end");
    state_stack.push_back("G");
    state_stack.push_back("begin");
    state_stack.push_back("D");
    state_stack.push_back("var");
    state_stack.push_back(";");
    state_stack.push_back("I");
    state_stack.push_back("program");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "LK") {
    state_stack.push_back("K");
    state_stack.push_back("L");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "JK") {
    state_stack.push_back("K");

```

```

    state_stack.push_back("J");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "B:C:") {
    state_stack.push_back(";");
    state_stack.push_back("C");
    state_stack.push_back(":");
    state_stack.push_back("B");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "IM") {
    state_stack.push_back("M");
    state_stack.push_back("I");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == ",IM") {
    state_stack.push_back("M");
    state_stack.push_back("I");
    state_stack.push_back(",");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "SU") {
    state_stack.push_back("U");
    state_stack.push_back("S");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "W") {
    state_stack.push_back("W");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "A") {
    state_stack.push_back("A");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "show(I);") {
    state_stack.push_back(";");
    state_stack.push_back("I");
    state_stack.push_back("(");
    state_stack.push_back("show");
    state = state_stack.back();
    state_stack.pop_back();
} else if (state == "I=E;") {
    state_stack.push_back(";");
    state_stack.push_back("E");
    state_stack.push_back("=");
    state_stack.push_back("I");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "TQ") {
    state_stack.push_back("Q");
    state_stack.push_back("T");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "+TQ") {
    state_stack.push_back("Q");

```

```

    state_stack.push_back("T");
    state_stack.push_back("+");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "-TQ") {
    state_stack.push_back("Q");
    state_stack.push_back("T");
    state_stack.push_back("-");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "FR") {
    state_stack.push_back("R");
    state_stack.push_back("F");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "*FR") {
    state_stack.push_back("R");
    state_stack.push_back("F");
    state_stack.push_back("*");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "/FR") {
    state_stack.push_back("R");
    state_stack.push_back("F");
    state_stack.push_back("/");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "I") {
    state_stack.push_back("I");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "N") {
    state_stack.push_back("N");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "(E)") {
    state_stack.push_back("(");
    state_stack.push_back("E");
    state_stack.push_back("(");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "JO") {
    state_stack.push_back("O");
    state_stack.push_back("J");
    state = state_stack.back();
    state_stack.pop_back();

} else if (state == "HJO") {
    state_stack.push_back("O");
    state_stack.push_back("J");
    state_stack.push_back("H");

```



```

state = state_stack.back();
state_stack.pop_back();

} else if (state == "lambda") {
state = state_stack.back();
state_stack.pop_back();

} else if (state == "a-e") {
if (iter >= "a" && iter <= "e") {
cout << "Successful Match: " << iter << endl;
state = state_stack.back();
state_stack.pop_back();

word_match = true;
}
} else if (state == "0-9") {
if (iter >= "0" && iter <= "9") {
cout << "Successful Match: " << iter << endl;
state = state_stack.back();
state_stack.pop_back();
word_match = true;
}
} else if (state == "null") { //finding errors and putting out error messages for states that were pushed on
the stack but not on the file
if(iter >= "a" && iter <= "e" && row == 5){
cout << ",tis missing\n";
break;
} else if(row == 14 && iter >= "a" && iter <= "e" && state >= "0" && state <= "9"){
cout << "Invalid variable\n";
break;
} else if(row == 14){
cout << ",tis missing\n";
break;
} else if (iter == "begin") {
cout << iter << "\tis expected\n";
break;
} else if (iter == "integer" && (row == 5)) {
cout << ".tis missing\n";
break;
} else if (row == 21 && iter == "=") {
cout << "Invalid Expression: Missing Variable\n";
break;
} else if(row == 21){
if(!check_spelling(iter)){
cout << "show\t is expected\n";

}
break;
} else if(row==6){
if(!check_spelling(iter)){
cout << "integer\t is expected\n";

}
break;
} else if(row==1){
cout << "Unknown Identifier\n";
break;
} else if(row == 7){
cout << "Invalid Expression:\tMissing Variable\n";

```

```

        break;
    }else if(row == 11){
        cout << "invalid Expression:\tMissing Expression\n";
        break;
    }
    break;
}
else if(state=="var" && iter != "var"){ //these errors are for states that weren't pushed on the stack and
made errors
    cout << state << "\tis expected\n";
    state = "null";
    break;
}
else if(state=="begin" && iter != "begin"){
    cout << state << "\tis expected\n";
    state = "null";
    break;
}
else if(state=="," && iter != ","){
    cout << state << "\tis missing\n";
    state = "null";
    break;
}
else if(state==":" && iter != ":"){
    cout << state << "\tis missing\n";
    state = "null";
    break;
}
else if(state=="=" && iter != "="){
    cout << state << "\tis missing\n";
    state = "null";
    break;
}
else if(state==";" && iter != ";"){
    cout << state << "\tis missing\n";
    state = "null";
    break;
}
else {
    state = "null";
    break;
}

}

return state_stack;
}

```

Sample Run on Correct Program:

```
/Users/Theresa/CLionProjects/untitled13/cmake-build-debug/untitled13
```

```
Previous finalp2.txt file found and deleted
```

```
Starting Part II.
```

```
<READING PROGRAM>
```

```
No error.
```

```
Accepted!
```

```
Creating new filep3.cpp
```

```
File filep3.cpp has been created.
```

```
Process finished with exit code 0
```

Sample Run Error ('program is expected')

```
/Users/Theresa/CLionProjects/untitled13/cmake-build-debug/untitled13
```

```
Previous finalp2.txt file found and deleted
```

```
Starting Part II.
```

```
program      is expected
```

```
Rejected.
```

```
Process finished with exit code 0
```