**Joshua Wilkes  
Programming Languages  
Project 2  
Functional Decomposition**

**User-defined data structures used as parameters in the functions**

struct lexemeNode

{

int line;

char word[30];

char whatAmI[15];

lexemeNode \*next;

};

struct errorNode

{

int line;

char message[75];

errorNode \*next;

};

struct declaredNode

{

int line;

char word[30];

declaredNode \*next;

};

struct bufferNode

{

char word[30];

bufferNode \*next;

bufferNode \*last;

}

struct registerNode

{

char word[20];

registerNode \*next;

registerNode \*last;

}

struct registers

{

char R[3];

bool used;

}

**Files and Functions in the Program**

/\* This file creates a linked list to store declared identifiers found in the given text file.

\* Author: Joshua Wilkes

\* Version: 1.0

\* Last Updated: 02/28/2019

\*/

**declaredIdentifiersList.h**

/\* Method to initially create the declared identifiers linked list \*/

**declaredNode\* createDIList(int line, char newWord[]);**

/\* Method to push new data into declared identifiers linked list \*/

**void pushDI(int line, char newWord[]);**

/\* Method to print all nodes and node data in declared identifiers linked list \*/

**void printDIList();**

/\* Returns head of declared identifiers linked list \*/

**errorNode\* getErrorHead();**

/\* This file creates a linked list to store errors found in the code.

\* Author: Joshua Wilkes

\* Version: 2.0

\* Last Updated: 02/28/2019

\*/

**errorList.h**

/\* Method to initially create the error linked list \*/

**errorNode\* createErrorList(int line, char message[]);**

/\* Method to push new data into error linked list \*/

**void pushErrors(int line, char message[]);**

/\* Method to print all nodes and node data in error linked list \*/

**void printErrorList();**

/\* Returns head of error linked list \*/

**errorNode\* getErrorHead();**

/\* This file analyzes a text file one character at a time, identifies tokens, and inserts the identified tokens into the linked list.

\* Author: Joshua Wilkes

\* Version: 2.0

\* Last Updated: 02/28/2019

\*/

**lexicalAnalyzer.h**

/\* Begins analysis of the file. To be called in the main method. \*/

**void beginAnalysis(FILE \*file);**

/\* A char is passed in from the file and when it is identified by the method, the rest of the token is gathered, identified, and passed into the linked list \*/

**int analyzeChar(char c);**

/\* Clears the string variable responsible for storing the token to be passed to the linked list \*/

**void clearLexeme(int length);**

/\* Checks a string against an array of reserved words. If not found, token is put into linked list as an identifier \*/

**bool isReservedWord(char word[]);**

/\* Checks a character against an array of operators. \*/

**bool isOperator(char c);**

/\* Checks a character against an array of delimiters. \*/

**bool isDelimiter(char c);**

/\* This file iterates through the token linked list and checks if the order of tokens/types is valid based on the rules set forth in the project.

\* Author: Joshua Wilkes

\* Version: 2.0

\* Last Updated: 02/28/2019

\*/

**parser.h**

/\* Returns result of parsing through program. \*/

**bool isValidProgram();**

/\* Skips all lines of comments during parsing \*/

**void skipComments();**

/\* Identifies a declaration statement and inserts valid identifies into declared identifiers linked list \*/

**declarationStatement();**

/\* Begins the process for identifying an assignment statement in program \*/

**void assignmentStatement();**

/\* Identifies an expression in an assignment statement \*/

**bool expression();**

/\* Identifies a term in an expression \*/

**bool term();**

/\* Identifies a factor in a term \*/

**bool factor();**

/\* This file is a buffer for the register stack to assist in creating a postfix expression.

\* Author: Joshua Wilkes

\* Version: 1.0

\* Last Updated: 02/28/2019

\*/

**registerBuffer.h**

/\* Method to initially create the buffer’s linked list \*/

**bufferNode\* createBufferStack(char word[]);**

/\* Method to push new data into buffer’s linked list \*/

**void pushToBuffer(char newWord[]);**

/\* Method to remove the last data item from buffer’s linked list \*/

**char \*popFromBuffer();**

/\* Method to remove a data item from the middle of the buffer’s linked list \*/

**void bufferRemoveFromMiddle(char \*word);**

/\* Method to print all buffer data in buffer’s linked list \*/

**void printBufferStack();**

/\* Returns head node of buffer’s linked list \*/

**lexemeNode\* getBufferHead();**

/\* Returns last node of buffer’s linked list \*/

**lexemeNode\* getBufferLast();**

/\* This file holds the finished postfix expression generated by the parser

\* Author: Joshua Wilkes

\* Version: 1.0

\* Last Updated: 02/28/2019

\*/

**registerStack.h**

/\* Method to initially create the register’s linked list \*/

**registerNode\* createRegisterStack(char word[]);**

/\* Method to push new data into register’s linked list \*/

**void pushToRegister(char newWord[]);**

/\* Method to print all nodes and node data in register’s linked list \*/

**void printRegisterStack();**

/\* Returns registerHead of register’s linked list \*/

**registerNode \*getRegisterHead();**

/\* This method takes the generated postfix expressions in the register and generates register based code to evaluate each expression, then it writes it all to a file based on the input file’s name \*/

**void writeOutRegister();**

/\* This file creates a linked list to store tokens found in the given text file.

\* Author: Joshua Wilkes

\* Version: 2.0

\* Last Updated: 02/28/2019

\*/

**tokenList.h**

/\* Method to initially create the token linked list \*/

**lexemeNode\* createTokenList(int line, char word[], char whatAmI[]);**

/\* Method to push new data into token linked list \*/

**void pushToken(int line, char newWord[], char whatAmI[]);**

/\* Method to print all nodes and node data in token linked list \*/

**void printTokenList();**

/\* Gets current identifier based on the iterator’s current position \*/

**char\* getCurrentIdentifier();**

/\* Returns head of token linked list \*/

**lexemeNode\* getTokenHead();**