**Alex Lindemann**

**Programming Languages Project 2 Functional Decomposition**

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

```
struct Entry{
        char value[SIZE];
         int type;
};
struct SymbolTable{
        Entry entries[SIZE];
         int size;
};
        struct Parameters {
        char *fileName;
        int currentRegister;
        FILE *inputFile;
         FILE *outputFile;
         int finishedDeclarations;
         int ch;
         int lineno;
         int pos;
         struct SymbolTable table;
        int lookahead;
        char value[SIZE];
        char postfix[SIZE];
};
typedef struct Parameters *paramsP;
```

## Files and Functions in the Program

/* The Recursive Descent Parser that checks the syntax and, to a degree, the semantics an input file. */

### Parser.c

```
/* This recognizes statements like a = 2 + 3 */
void AssignStmt(paramsP);
```

/* Functionality for things like x + 3; */

**void expression(paramsP);**

/* For things like 2*3 */

**void term(paramsP);**

/* For doing things with parentheses. This makes sure we have the same amount of open parentheticals as close. */

**void factor(paramsP);**

/* This makes sure we get our expected things like BEGIN, END, semicolons and such. If we expect something and it's not there, we quit. */

**void match(int, paramsP);**

/* the function that sets it all in motion. matches BEGIN and END because a program must have those things at the very least. */

**void parse( paramsP);**

/* Prints the Symbol Table*/

**void listIdentifiers(paramsP);**

/* Checks for illegal tokens after the "end" keyword. Allows for comments. If it finds any type of white space or a comment, it calls itself to advance lookahead. If it finds anything other than whitespace or a comment it gives an error. */

**void checkAfterEnd(paramsP);**

/* Lexical Analyzer that extracts tokens for the parser. */

**Lexan.c**

/* Gets chars from the inputFile and gives them to the parser.*/

**int lexan(paramsP);**

/* Checks to find the given token value in the Symbol Table*/

**int find(char *, SymbolTable);**

/* inserts a string token into the Symbol Table */

**void addValueToSymbolTable(char *, paramsP);**

/*builds the symbol table and params with default values*/

**void buildTable(params, char *);**

/* A constructor of sorts that returns a pointer to a symbol table, char, lookahead, etc.*/

**paramsP newParams(char *);**

/* Returns TRUE (1) if the identifier is legal, FALSE (0) if it was not. Identifiers are legal if it does not end in an '_' and it does not have two consecutive '_'s */

**int legalIdentifier(char *, paramsP);**

/* Inserts the buffer of int decls into the symbol table*/

**void getBufferIntoTable(char *, paramsP);**

/*This reads in the declarations and has the functionality for detecting a missing semicolon.*/

**void readInDecls(char *, paramsP);**

/*Function for quitting the program if there's an undeclared variable.*/

**void undeclaredError(paramsP );**

/* this reads in identifiers.*/

**void readInID(paramsP);**

/*changes changes the extension on the file name to ".out" */

**void changeExtension(char *);**

/* Creates a new params and immediately starts parsing the file);

**main.c**