Mikayla Timm

Programming Languages

Project 2

Functional Decomposition

**User-defined data structures**

*//Symbol struct – used for holding the value of each identifier/begin/end for storing in the table*

**struct** Symbol{

**char** value[**MAXARRAYSIZE**];

**int** type;

**int** pos;

};

*//SymbolTable struct – has an array of symbols and a size. Used for seeing if variables are declared, storing variables, and storing begin/end symbols*

**struct** SymbolTable{

SymbolP symbolPs[**MAXARRAYSIZE**]; *//array of symbolPs*

**int** size;

};

*//data struct holding all the info for all the operations*

**struct** Data{

**char** \*comma; *//just holds a comma and a null char so I can easily concat for postfix*

SymbolTableP table; *//symbol table*

**int** lookahead;

**int** lineno;

FILE \*file; *//input file*

**int** ch;

**int** pos;

**char** identifier[**MAXARRAYSIZE**];

**int** declsDone;

**int** currentReg;

**char** assignVar[**MAXARRAYSIZE**];

**char** postfixExpr[**MAXARRAYSIZE**];

**char** fileName[**MAXARRAYSIZE**]; *//name of output file*

FILE \*outFile;

};

**Files and Functions in the Program**

*/\*  
\* This file outlines the functions used by the lexical analyzer and*

*\* defines the struct used to hold all of the Data necessary to perform the operations,*

*\* including the file, lookahead, current char (ch), Symbol table, and symbols.  
\*/*

Lexan.c

*/\**

*\* Set up the struct to hold all of the information used by the whole program in the Data struct.*

*\* Initialize variables.*

*\*/*

DataP createData(FILE \*file);

*/\**

*\* Create the symbol table and set up the 1st and 2nd elements for begin and end.*

*\* Return the table to the data struct.*

*\*/*

SymbolTableP createTable();

*/\**

*\* Finds an identifier value in the symbol table held in the data struct.*

*\* If not found, return NOT\_FOUND value. Else return the index where the ID was found.*

*\*/*

**int** find(**char** \*value, DataP data);

*/\**

*\* Inserts an identifier into the symbol table held in the data struct.*

*\*/*

**void** insert(**char** \*value, DataP data);

*/\**

*\* Lexan function that processes all of the characters in the program, deciding if*

*\* it's an ID, NUM, comment, new line or space or tab, or operator.*

*\* Taken from Dr. Coffey's pseudocode. Returns the character to the parser.*

*\*/*

**int** lexan(DataP data);

*/\**

*\* Reads in all of the characters for each ID (alphas, digits, and \_s).*

*\*/*

**void** readID(DataP data);

*/\**

*\* Checks the ID just read in to see if it's legal, as in no consecutive \_s and*

*\* not ending in an \_.*

*\*/*

**int** checkIfIDLegal(DataP data);

*/\**

*\* Reads in the int IDs until a semicolon is reached.*

*\*/*

**void** getInts(DataP data);

*/\**

*\* Reads until a , or ; is reached*

*\*/*

**void** getToEnd(DataP data);

*/\**

*\* Reads until next ID*

*\*/*

**void** getToID(DataP data);

*/\**

*\* Reads in the number and saves it into identifier*

*\*/*

**void** readNum(DataP data);  
*/\**

*\* exits the program. closes the files*

*\*/*

**void** stopProgram(DataP data);

*/\*  
\* This file implements all of the functions*

*\* for the parser. Most of this was taken from Dr. Coffey's pseudocode.*

*\* Exits the program if there is an error.  
\*/*

Parser.c

*/\**

*\* Taken from Dr. Coffey's pseudocode.*

*\* Checks lexan for an ID, calls the expression function to get an assignment statement from the program.*

*\*/*

**void** AssignStmt(DataP data);

*/\**

*\* evaluates expressions by checking term, operator, term. Can be recursively called for nested expressions in factor.*

*\*/*

**void** expression(DataP data);

*/\**

*\* Evaluates a "term" by calling factor and making sure the term is made up of either*

*\* just an ID or NUM, OR an ID/NUM \* or / an ID/NUM, or a nested term with parens.*

*\*/*

**void** term(DataP data);

*/\**

*\* "Factors" the statement by either just confirming there is an ID or NUM where expected, or*

*\* if there is a nested expression () recursively calls expression and matches closing paren.*

*\*/*

**void** factor(DataP data);

*/\**

*\* Calls lexan to see if character/ID/NUM read in matches "t" passed in. If not, prints an error and closes program.*

*\*/*

**void** match(**int** t,DataP data);

*/\**

*\* Parses the statements in the program. Reads through until end of file is reached, making calls to other functions*

*\* to check if all statements are legal. If so, it lists all the IDs in the symbol table. If not, exits with error message.*

*\*/*

**void** parse(DataP data);

*/\**

*\* Removes the final comma from the postfix expression after it's all done*

*\*/*

**void** removeFinalComma(DataP data);

*/\*  
\* This file opens the file for reading and calls the parser to analyze the program.  
/\**

RecursiveDescent.c

*/\**

*\* Get the filename of the file passed in without the extension*

*\*/*

**void** getFileName(DataP data, **char** \*arg);

*/\**

*\* Opens the file for reading and calls the parser to analyze the program.*

*\*/*

**int** main(**int** argc, **char** \*argv[])