**Probal Chandra Dhar**

**Advanced Programming Languages**

**Project 01**

**Functional Decomposition**

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

/\* This is the structure used to handle symbol table using externally chained hash table \*/

/\* lexemeT - array contains the token itself \*/

/\* tokenType - value of the token \*/

/\* entry \*next - pointer to the next record of the symbol table \*/

**struct entry {**

**char lexemeT[STRMAX];**

**int tokenType;**

**struct entry \*next;**

**};**

// SYMBOL table

**struct entry \*symtable[SYMMAX];**

in **symbol.c**

**Files and Functions in the Program**

/\* This is the main file for the project 1, it opens the file to read and \*/

/\* calls init() which add some lexeme in the symbol table \*/

/\* and then it calls the parse() function to parse the program written in the file \*/

/\* it also includes global.h for the default values used in other files \*/

**main.c**

/\* This file deals with symbol table, it has a entry structure \*/

/\* functions: lookup(), insert(), hashFunction(), getLexeme(), getTokenType() etc \*/

/\* these functions to use when we need to insert any lexeme in the symbol table \*/

/\* and also to lookup any tokens from the symbol table \*/

**symbol.c**

/\* init() function assigns NULL to every record of the symbol table \*/

/\* and insert the keywords in the symbol table \*/

**init();**

/\* lookup() function search any tokens in the symbol table \*/

/\* it accepts a string as lexeme and search through the symbol table \*/

/\* it returns FOUND or NOT\_FOUND depending on the search of the lexeme \*/

**int lookup(char s[]);**

/\* HASH FUNCTION - summing up all character's ascii value \*/

/\* it receives the lexeme as parameter \*/

/\* it returns the hash value for the string \*/

**int hashFunction ( char lexemeP[] );**

/\* Function to create a node for the entry sturcture to insert the node in the symbol table \*/

/\* it receives token and tokenval as parameter \*/

/\* this function returns the full node of type \*entry \*/

**struct entry \* createEntry (char \*name, int token );**

/\* insert() function insert the lexeme in the symbol table using the hash value for that string \*/

/\* it receives token and tokenType as parameter \*/

/\* it returns the hashIndex for the string \*/

**int insert ( char s[], int tok );**

/\* This function returns the current entry lexeme that points \*/

/\* to the recent inserted or lookedup lexeme in the symbol table \*/

/\* this pointer is set up in the insert() and in lookup() \*/

**char \* getLexeme();**

/\* tokenType() function returns the tokenType of the recent token that has been \*/

/\* inserted or lookedup in the symbol table \*/

/\* current pointer is being set in the insert() and lookup() functions \*/

**int getTokenType();**

/\* This file checks for each character in the input and categorize them and return values on the basis on them \*/

/\* it also have function to open a file \*/

**lexer.c**

/\* openFile() function opens the file to be read for parsing \*/

/\* this function don't receive any parameter and return nothing \*/

**void openFile(char \*fileName);**

/\* closes the opened file \*/

**void closeFile();**

/\* lexan() function checks each character in the input from the file \*/

/\* it defines if the lexeme is NUM, blank space or any valid token type \*/

/\* this function also lookup & insert any valid token in the symbol table \*/

**int lexan();**

/\* This file generate an error message and exit the program \*/

**error.c**

/\* This function print error message with the line number in the display and exit the program \*/

**error(char \* m);**

/\* This file contains the global values for reserved keywords, Identifier, Number, IF, WHILE etc. \*/

/\* also some default values for the return used in other files \*/

**global.h**

/\* This file emits the opeartors and prints them in the screen \*/

**emitter.c**

/\* emit() function receives t and tval which is token and tokenval \*/

/\* depending on the token and tokenval something would print in the screen \*/

**emit(int t, int tval);**

/\* Parser that component that breaks data into smaller elements for easy translation \*/

/\* it takes lexeme from lexer.c and check if it's a valid token or not \*/

**parser.c**

/\* parse() function takes the lookahead and check what function I should call \*/

/\* depending on the lookahead it calles for the matches specific keywords \*/

/\* parse() normally calls expr() to check the statements & expression \*/

**parse();**

/\* statement() function check the next lexeme and match that with the current lookahead \*/

/\* it tries to match the lookahead with interger declaration, selector, iterator or expression \*/

**statement();**

/\* this function expect any IF..ENDIF selector statement. It matches with the grammar \*/

/\* and stops the program to be execute if it's don't match with the grammar

\*/

**selector();**

/\* this function executes when the lexeme is for iterator, for this program it only WHILE...ENDWHILE \*/

/\* it matches with the grammar and stops if the program has any invalid grammar or lexeme \*/

**iterator();**

/\* expr() function checks the statement and search if it's contain any operator \*/

/\* before checking for operator expr() is calling term() \*/

/\* the grammar for it should be <expr> ::= <term>{[+|-]<term>}\* \*/

/\* so a expr may have one or more term \*/

**expr();**

/\* term() function checks for any factor in the statement and calls factor() \*/

/\* the grammar for the term would be <term> ::= <factor>{[\*|/]<factor>}\* \*/

/\* so it checks for one or more factors in the statement \*/

**term();**

/\* factor() function checks for any variable, number, any comparison operator, interger decelaration etc in a factor \*/

/\* the grammar for any factor would be <factor> ::= <variable> | <number> | (<expr>) \*/

/\* so it's either a variable, number, ID, COMPAR etc. \*/

**factor();**

/\* match() function match the lexeme with the expected lookahead \*/

/\* when it matches it calls lexan() and store the lexeme in lookahead \*/

/\* otherwise it calls for error() which exits the program \*/

**int match(int t);**