Eric Olechovski

Kyle Wilson

COSC 424

Dr. Fries

Implementation of the Lexical Analyzer

We used a total of 7 classes to implement the lexical analyzer including: a class for the lexical analyzer that contained a main method and opened the file to be analyzed, a lexical analyzer reader that read a file and performed the lexical analysis, a constants class, a symbol table class, a token class, an interface called dictionary that represented a hashtable, and a hashtable class that would be used to represent the symbol table. Classes were organized as business objects, input/output objects, and utility objects.

The symbol table mechanism behind the scenes was a hashtable. A hashtable was used to map keys to values, or in our case lexemes with an id. In the symbol table class, a hashtable was created as an instance variable taking type arguments of String and Integer. The string represented the lexeme being added to the table, and the Integer represented the constant value for an id. Additionally, we chose to use a hashtable because duplicate keys cannot exist in the table meaning there was no need to check if a key already had existed. The installID method in the symbol table returned the index of the lexeme in the table;

To handle constants, we made a separate class that would contain predefined variables to represent operators and keywords. In our constants class, we represented ids, numbers, semicolons, commas, control structures, arithmetic operators, relational operators, assignment operators, and keywords. Each category also had a variable defined, such as addop or multop, to represent the category. We chose not to define variables to represent categories for semicolons, commas, and control structures, and instead made a separate constructor in the token class that only took a value to construct a token object. To access the values assigned in the constants class, the class would be imported and used in statements such as CONSTANT.ID to access the value.

Table Summarization

|  |  |
| --- | --- |
| ID | 100 |
| NUM | 102 |
| SEMICOLON | 103 |
| COMMA | 104 |
| LEFT PAREN | 106 |
| RIGHT PAREN | 107 |
| LEFT BRACE | 108 |
| RIGHT BRACE | 109 |
| ADDITION OPERATOR | 200 |
| ADD | 201 |
| SUBTRACTION | 202 |
| OR | 203 |
| AND | 204 |
| MUTLIPLICATION OPERATOR | 300 |
| MULTIPLICATION | 301 |
| DIVISION | 302 |
| MODULUS | 303 |
| RELATIONAL OPERATOR | 400 |
| EQUALS EQUALS | 401 |
| NOT EQUALS | 402 |
| LESS THAN | 403 |
| LESS THAN OR EQUALS TO | 404 |
| GREATER THAN | 405 |
| GREATER THAN OR EQUALS TO | 407 |
| NOT | 407 |
| ASSIGN OPERATOR | 500 |
| EQUALS | 501 |
| KEYWORD | 600 |
| VOID | 601 |
| INT | 602 |
| DOUBLE | 603 |
| CHAR | 604 |
| IF | 605 |
| ELSE | 606 |
| WHILE | 607 |
| COMMENT | 700 |
| SLASH\_STAR\_COMMENT | 701 |