**Assignment 2 Syntactic Analyzer**

**Problem Statement**

The problem is to determine whether the given source code is syntactically correct or not. We solve this using a top-down parser for syntactic analysis. If the source code’s syntax is correct, the parse trace will be outputed to a file. If there is an error, the details of the first error encountered will be printed to the output file.

**How to use the Program**

To execute the program:

Windows (Visual Studio)

1. RUN compiler.cpp in VS or compiler.exe
2. Enter the .txt file name if it’s in the current directory, or the relative path if it’s not.

Linux

1. Within command prompt, open ‘compiler’ directory.
2. Compile it with the command “g++ compiler.cpp -o compiler”.
3. Run it with “./compiler”
4. When the program runs it will prompt you to enter the source file name’s relative path (ex. ../test.txt).

**Design of the Program**

Algorithm Design:

Our implementation is the Predictive Recursive Descent Parser. This algorithm requires removing left recursions and left factorization from the grammar productions and using First and Follow sets to eliminate backtracking.

For our data structures, arrays are used for the First and Follow sets. We have defined a Token type. It tells what line the token was found on, its type, lexeme, and a comment string for comments such as the associated error.

RAT20SU Grammar Productions:

|  |
| --- |
| <RAT20SU> ::= $$ <OPT\_DEC\_LIST> <STATEMENT\_LIST> $$  <OPT\_DEC\_LIST> ::= <DECLARATION\_LIST> | <EMPTY>  <DECLARATION\_LIST> ::= <DECLARATION> ; <DECLARATION\_LIST\_PRIME>  <DECLARATION\_LIST\_PRIME > ::= <DECLARATION\_LIST> | <EMPTY>  <DECLARATION> ::= <QUALIFIER> <TT\_Identifier>  <QUALIFIER> ::= integer | boolean  <STATEMENT\_LIST> ::= <STATEMENT> <STATEMENT\_LIST\_PRIME>  <STATEMENT\_LIST\_PRIME> ::= <STATEMENT\_LIST> | <EMPTY>  <STATEMENT> ::= <COMPOUND> | <ASSIGN> | <IF> | <GET> | <PUT>  | <WHILE>  <COMPOUND> ::= { <STATEMENT> <STATEMENT\_LIST\_PRIME>}  <ASSIGN> ::= <TT\_Identifier> = <EXPRESSION> ;  <EXPRESSION> ::= <TERM> <EXPRESSION\_PRIME>  <EXPRESSION\_PRIME> ::= + <TERM> <EXPRESSION\_PRIME>  | - <TERM> <EXPRESSION\_PRIME>  | <EMPTY>  <TERM> ::= <FACTOR> <TERM\_PRIME>  <TERM\_PRIME> ::= \* <FACTOR> <TERM\_PRIME>  | / <FACTOR> <TERM\_PRIME>  | <EMPTY>  <FACTOR> ::= - <PRIMARY> | <PRIMARY>  <PRIMARY> ::= <TT\_Identifier>  | <TT\_Integer>  | (<FACTOR> <TERM\_PRIME> <EXPRESSION\_PRIME>)  | true  | false  <WHILE> ::= while ( <CONDITION> ) <STATEMENT>  <CONDITION> ::= <EXPRESSION> <RELOP> <EXPRESSION>  <RELOP> ::= == | > | <  <IF> ::= if ( <CONDITION> ) <STATEMENT> <IF\_PRIME>  <IF\_PRIME> ::= fi | otherwise <STATEMENT> fi  <PUT> ::= put ( <TT\_Identifier> ) ; // means print  <GET> ::= get ( <TT\_Identifier> ) ; // like scanf  <EMPTY> ::= ɛ |

|  |  |  |
| --- | --- | --- |
| **RULE** | **FIRSTS** | **FOLLOWS** |
| RAT20SU | $$ | $ |
| OPT\_DEC\_LIST | ε, integer, boolean | {, TT\_Identifier, if, get, put, while |
| DECLARATION\_LIST | integer, boolean | {, TT\_Identifier, if, get, put, while |
| DECLARATION\_LIST\_PRIME | ε, integer, boolean | {, TT\_Identifier, if, get, put, while |
| DECLARATION | integer, boolean | ; |
| QUALIFIER | integer, boolean | TT\_Identifier |
| STATEMENT\_LIST | {, TT\_Identifier, if, get, put, while | $$, } |
| STATEMENT\_LIST\_PRIME | ε, {, TT\_Identifier, if, get, put, while | $$, } |
| STATEMENT | {, TT\_Identifier, if, get, put, while | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| COMPOUND | { | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| ASSIGN | TT\_Identifier | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| IF | if | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| IF\_PRIME | fi, otherwise | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| PUT | put | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| GET | get | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| WHILE | while | {, TT\_Identifier, if, get, put, while, $$, fi, otherwise, } |
| CONDITION | -, TT\_Identifier, TT\_Integer, (, true, false | ) |
| RELOP | ==, >, < | -, TT\_Identifier, TT\_Integer, (, true, false |
| EXPRESSION | -, TT\_Identifier, TT\_Integer, (, true, false | ;, ==, >, <, ) |
| EXPRESSION\_PRIME | ε, +, - | ;, ==, >, <, ) |
| TERM | -, TT\_Identifier, TT\_Integer, (, true, false | +, -, ;, ==, >, <, ) |
| TERM\_PRIME | ε, \*, / | +, -, ;, ==, >, <, ) |
| FACTOR | -, TT\_Identifier, TT\_Integer, (, true, false | \*, /, +, -, ;, ==, >, <, ) |
| PRIMARY | TT\_Identifier, TT\_Integer, (, true, false | \*, /, +, -, ;, ==, >, <, ) |
| EMPTY | ε | +, -, ;, ==, >, <, ), $$, {, }, TT\_Identifier, if, get, put, while |

**Limitations:**

None

**Shortcomings**

None