COMPILER DESIGN TUTORIAL

IMPLEMENTATION OF SCANNER, RECURSSIVE DESCENT PARSER AND A SYMBOL TABLE

BY:

DHARITHRI M 4NI18CS121
PRANAV B 4NI18CS122
VENU D 4NI18CS126

program $\Longrightarrow$ global declList stmtList end
declList $\Longrightarrow$ decl declList | ϵ
decl $\Longrightarrow$ def typeList end
typeList $\Longrightarrow$ varList : type ; typeListA
typeListA $\Longrightarrow$ varList : type ; typeListA | ϵ
varList $\Longrightarrow$ var , varList | var
var $\Longrightarrow$ identifier
type $\Longrightarrow$ int | char
stmtList $\Longrightarrow$ stmt ; stmtListA
stmtListA $\Longrightarrow$ stmt ; stmtListA | ϵ
stmt $\Longrightarrow$ assignmentStmt | readStmt | printStmt | ifStmt | whileStmt
assignmentStmt $\Longrightarrow$ identifier = exp
readStmt $\Longrightarrow$ read identifier
printStmt $\Longrightarrow$ print exp
ifStmt $\Longrightarrow$ if bexp : stmtList elsePart end
elsePart $\Longrightarrow$ else stmtList | ϵ
whileStmt $\Longrightarrow$ while bexp : stmtList *end*
bexp $\Longrightarrow$ (bexp) | exp < exp bexpA | exp == exp bexpA | not bexp bexpA
bexpA $\Longrightarrow$ or bexp bexpA | and bexp bexpA | ϵ
exp $\Longrightarrow$ identifier expA | number expA | (exp)
expA $\Longrightarrow$ - exp expA | + exp expA | * exp expA | / exp expA | ϵ
number $\Longrightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Production rules

Modified production rules after removing the left recursion and introducing new productions.

01

SCANNER

02

RECURSSIVE DESCENT PARSER

03

SYMBOL TABLE

Implemented with C++ 17

Steps involved in scanning

1

Read source file character by character

2

Check if character is valid and add to a string

3

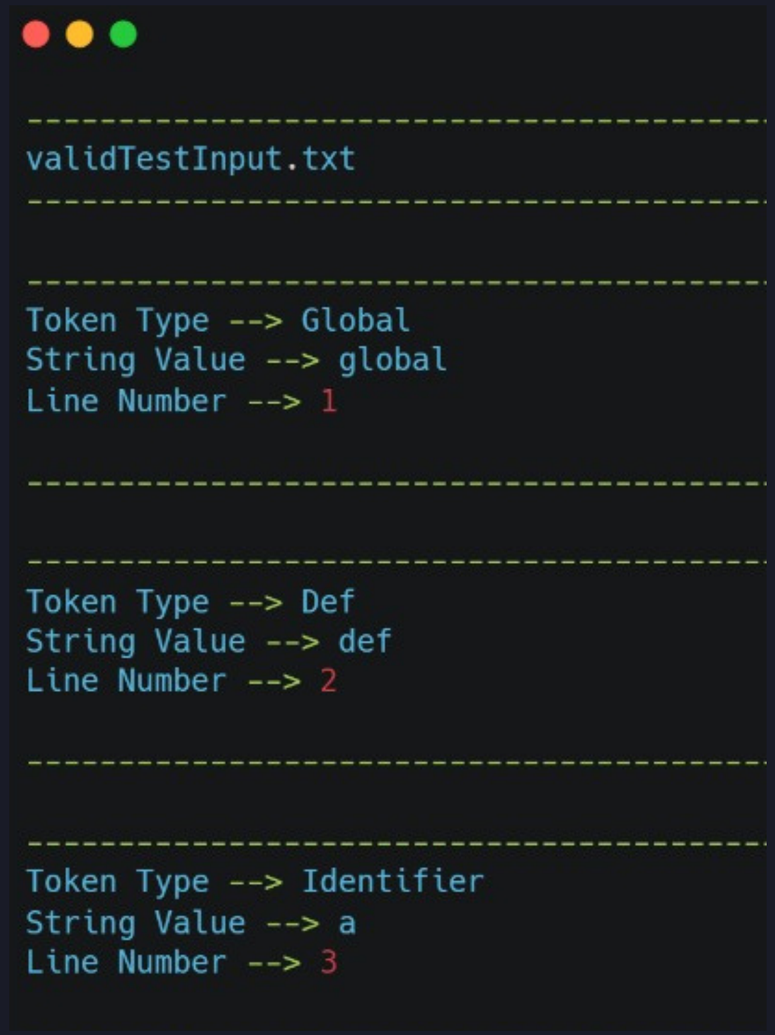
Classify the string's token type

4

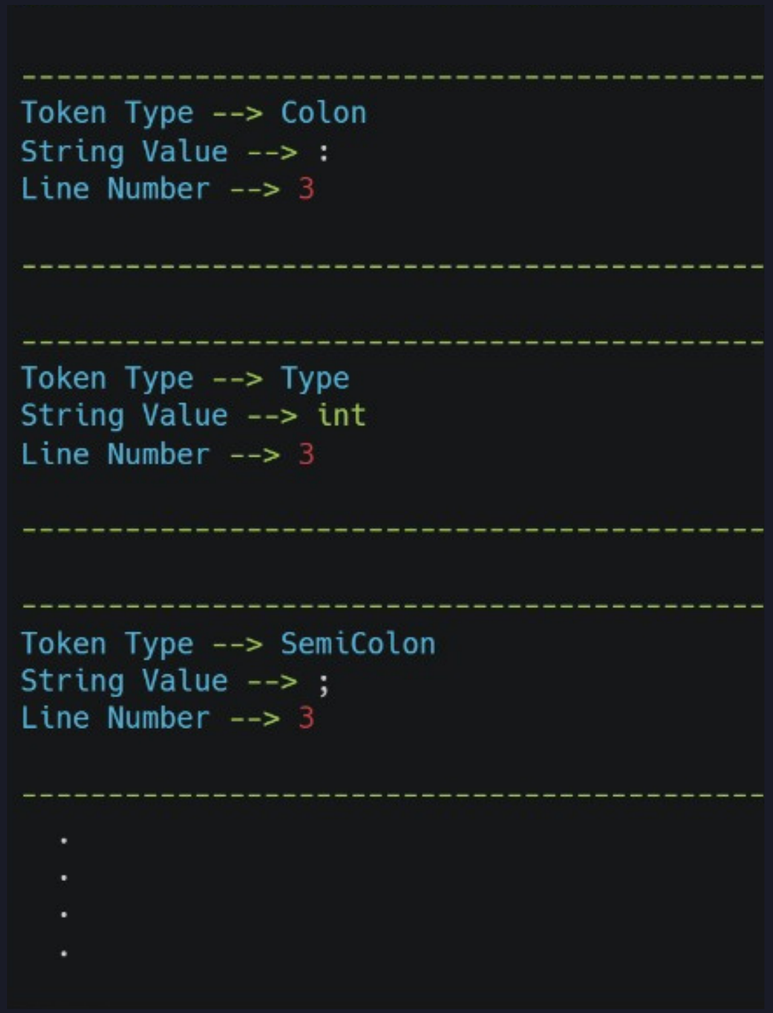
Return a token record containing token type, value and additional attributes.

```
class Scanner {

    private:
        char currentChar;
        TokenType currentKind;
        std::string currentString;
        std::fstream inFile;
        static int line;
        // methods
        void acceptChar();
        void discardChar();
        TokenType scanToken();
        void scanSeparator();


    public:
        Scanner(std::string filename);
        ~Scanner();
        // methods
        TokenRecord getToken();

        // getters
        char getCurrentChar();

};
```

Example output for scanner



```
----------------------------------------
validTestInput.txt
----------------------------------------

----------------------------------------
Token Type --> Global
String Value --> global
Line Number --> 1

----------------------------------------

----------------------------------------
Token Type --> Def
String Value --> def
Line Number --> 2

----------------------------------------

----------------------------------------
Token Type --> Identifier
String Value --> a
Line Number --> 3
```



```
----------------------------------------
Token Type --> Colon
String Value --> :
Line Number --> 3

----------------------------------------

----------------------------------------
Token Type --> Type
String Value --> int
Line Number --> 3

----------------------------------------

----------------------------------------
Token Type --> SemiColon
String Value --> ;
Line Number --> 3

----------------------------------------
  .
  .
  .
  .
```

Steps involved in parsing

1

Get initial token and apply production rules for start symbol.

2

Start symbol production calls function to apply productions for other rules.

3

Recursively check if the token is valid or not while continuously getting the next token.

4

If all productions pass, check if there are no more tokens available. If not, source code is valid.

```
class Parser{
    private:
        TokenRecord currentToken;
        Scanner *scanner;
        SymbolTable symtab;
        int count;
        bool identifierCheckFlag;

        std::vector<std::string> symbolNamesList;
        //parsing methods
        void accept(TokenType expected);
        void acceptToken();
        bool checkToken(TokenType expected);
        void parseProgram();
        void parseDeclList();
        void parseDecl();
        void parseTypeList();
        void parseTypeListA();
        void parseType();
        void parseVarList();
        void parseVar();
        void parseStmtList();
        void parseStmtListA();
        void parseStmt();
        void parseAssignementStmt();
        void parseReadStmt();
        void parsePrintStmt();
        void parseIfStmt();
        void parseElsePart();
        void parseWhileStmt();
        void parseBexp();
        void parseBexpA();
        void parseExp();
        void parseExpA();
        void parseNumber();

    public:
        Parser(std::string fileName);
        void parse();
        void showSymbolTable();
        ~Parser();

};
```

```
class SymbolTable{

    private:
        std::map<std::string,SymTabRow> SymbolTableMap;

    public:
        void addSymbolToTable(std::string symbolName,int lineNumber ,std::string type);
        void setSymbolValue(std::string symbolName,std::string value);
        bool checkIfSymbolExists(std::string symbolName);
        void showSymbolTable();

};
```

Steps involved in Symbol Table

1

Initialize an empty hash map of a string symbol name and a symbol table row.

2

When parsing, if a symbol is found, add an entry to the table under that particular symbol name.

3

Each row contains symbol name, symbol type and a line number.

4

after successful parsing, symbol table is displayed.

Additional data structures used

```
struct SymTabRow {
    std::string symbolName;
    std::string type;
    std::string attribute;
    int lineNumber;

    friend std::ostream& operator << (std::ostream& out,const SymTabRow& obj);
};
```

Symbol Table Row.

Token Record.

```
struct TokenRecord{
    TokenType tokenType;
    std::string Value;
    int lineNumber;
    friend std::ostream& operator << (std::ostream& out,const TokenRecord& obj);
};
```

```
global
    def
        a:int;
        c,b:int;
    end
    a = 1;
    b = 7;
    if a == b :
        print a;
    else
        print b;
    end;
    while a < 5:
        b = b * 2;
        a = a + 10;
    end;
    print b;
end
```

Example of valid code for the given grammar

Output for the previous example

```
validTestInput.txt
Accepted token Global in line 1 as global
Accepted token Def in line 2 as def
Accepted token Identifier in line 3 as a
Accepted token Colon in line 3 as :
Accepted token Type in line 3 as int
Accepted token SemiColon in line 3 as ;
Accepted token Identifier in line 4 as c
Accepted token Comma in line 4 as ,
Accepted token Identifier in line 4 as b
Accepted token Colon in line 4 as :
Accepted token Type in line 4 as int
Accepted token SemiColon in line 4 as ;
Accepted token End in line 5 as end
Accepted token Identifier in line 6 as a
Accepted token Assignment in line 6 as =
Accepted token Number in line 6 as 1
Accepted token SemiColon in line 6 as ;
Accepted token Identifier in line 7 as b
Accepted token Assignment in line 7 as =
Accepted token Number in line 7 as 7
Accepted token SemiColon in line 7 as ;
Accepted token If in line 8 as if
Accepted token Identifier in line 8 as a
Accepted token Conditional in line 8 as ==
Accepted token Identifier in line 8 as b
Accepted token Colon in line 8 as :
Accepted token Print in line 9 as print
Accepted token Identifier in line 9 as a
Accepted token SemiColon in line 9 as ;
Accepted token Else in line 10 as else
Accepted token Print in line 11 as print
Accepted token Identifier in line 11 as b
Accepted token SemiColon in line 11 as ;
Accepted token End in line 12 as end
Accepted token SemiColon in line 12 as ;
Accepted token While in line 13 as while
Accepted token Identifier in line 13 as a
Accepted token Conditional in line 13 as <
Accepted token Number in line 13 as 5
Accepted token Colon in line 13 as :
Accepted token Identifier in line 14 as b
Accepted token Assignment in line 14 as =
Accepted token Identifier in line 14 as b
Accepted token Operator in line 14 as *
Accepted token Number in line 14 as 2
Accepted token SemiColon in line 14 as ;
Accepted token Identifier in line 15 as a
Accepted token Assignment in line 15 as =
Accepted token Identifier in line 15 as a
Accepted token Operator in line 15 as +
Accepted token Number in line 15 as 10
Accepted token SemiColon in line 15 as ;
Accepted token End in line 16 as end
Accepted token SemiColon in line 16 as ;
Accepted token Print in line 17 as print
Accepted token Identifier in line 17 as b
Accepted token SemiColon in line 17 as ;
Accepted token End in line 18 as end
VALID CODE

                    SYMBOL TABLE

    Symbol Name          Type          Line Number

            a            int                 3
            b            int                 4
            c            int                 4
```

```
global
    def
        a:int;
        b:int;
    end
    a = 1;
    b = 7;
    if a =<= b :
        print a;
    else
        print c;
    end;
    while a < 5:
        b = b * 2;
    end;
    print b;
end
```

Example of invalid code for the given grammar

Error is on line 8 with invalid operator (=<=)

```
--------------------------------------------------------------------------------
invalidTestInput.txt
--------------------------------------------------------------------------------
Accepted token Global in line 1 as global
Accepted token Def in line 2 as def
Accepted token Identifier in line 3 as a
Accepted token Colon in line 3 as :
Accepted token Type in line 3 as int
Accepted token SemiColon in line 3 as ;
Accepted token Identifier in line 4 as b
Accepted token Colon in line 4 as :
Accepted token Type in line 4 as int
Accepted token SemiColon in line 4 as ;
Accepted token End in line 5 as end
Accepted token Identifier in line 6 as a
Accepted token Assignment in line 6 as =
Accepted token Number in line 6 as 1
Accepted token SemiColon in line 6 as ;
Accepted token Identifier in line 7 as b
Accepted token Assignment in line 7 as =
Accepted token Number in line 7 as 7
Accepted token SemiColon in line 7 as ;
Accepted token If in line 8 as if
Accepted token Identifier in line 8 as a
INVALID
Syntax error on line 8  CURRENT TOKEN IS None  CURRENT TOKEN NAME IS =<= BUT EXPECTED TOKEN IS
Conditional
```

Output for the
previous invalid example

Thank you!