Raccoons : DSL for CSV Data Processing

Team members:-
Mane Pooja Vinod:- CS22BTECH11035 (Project Manager)
Deva Suvedh:- CS22BTECH11016 (System Architect)
Medikonda Sreekar:- CS22BTECH11037 (System Architect)
Bolla Nehasree:- CS22BTECH11012 (Language Guru)
C Sree Vyshnavi:- CS24RESCH11010 (Language Guru)
Surbhi:- CS22BTECH11057 (System Integrator)
Simhadri Nayak Ramavath:-CS22BTECH11049 (Tester)

LEXICAL ANALYZER

The lexical analyzer scans the input .rc files and categorizes sequences of characters into tokens based on predefined patterns. Here's a breakdown of its key components:

1. Tokens:-

There are several tokens like keywords, operators, punctuation and special types of tokens.

- Keywords:- for, while, if, else, int, float, string, bool, true, false, input, output, print,continue,break.
- Operators:- +=, -=, *=, /=, %=, relational operators like <=, >=, ==, !=, + , - , * , \ , % , = , < , > , & , ^ , | and logical operators like &&, || , ++ , –- , >> , << .
- **Punctuation**: ; , , , : , (,) , [,] , { , } , . . . ,”,’ .
- **Special Types**: EXPONENTIAL , PERCENTAGE , INTEGER , FLOAT , STRING , CSVFILE .

2. Regular Expressions for Token Matching:-

The %% block contains regular expressions that match various components of the input source code and map them to specific token types. For example:

- "for":- Matches the keyword "for" and returns the token FOR.
- [0-9]+(\.[0-9]+)?% :- Matches percentage numbers and returns the PERCENTAGE token.
- "\"([^\"\\]|\\.)*\"" :- Matches string literals enclosed in double quotes and returns the STRING token.
- [A-Za-z_]+\.csv :- Matches filenames with the .csv extension and returns the CSVFILE token.

3. File I/O for Output:-

- The lexer reads input from a file specified by the user and writes the tokenized output to an output file named output.txt.
- It uses yyin for reading from the input file and writes the lexed tokens to outfile.

4. Multi-line and Single-line Comment Handling:-

- **Single-line comments**: Lines starting with // are ignored by the lexer.
- **Multi-line comments**: Handled by matching the start (/*) and end (*/) sequences and skipping everything in between.

5. Exponential Numbers, Identifiers, and Dust:-

- The lexer is designed to recognize exponential numbers by the pattern [0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+).
- Identifiers follow the pattern [A-Za-z][A-Za-z0-9_]*.
- The token DUST is returned for any unrecognized characters that don't fit other patterns.

6. Count( ) Function:-

This function counts the columns and keeps track of the current position in the input file, making sure to handle new lines and other characters correctly.

7. End-of-File Handling:-

The **yywrap()** function ensures that the lexer returns 1 when it reaches to the end of the input, indicating that tokenization is complete.

8.Main Function:-

The main() function reads the input file name, opens the input and output files, and continuously calls yylex() to process tokens until the end of the input is reached.

SAMPLE I/O OF LEXICAL ANALYZER

INPUT OF sample1.rc:-

```
//reads csv file as input
input weather_data.csv;
// Converts CSV file into data frame
df = read('weather_data.csv');
// Fills missing values with the median
df_filled = df.miss_value(fill,df.median());
// Calculate aggregate statistics
mean_temp = df_filled['Temperature'].mean();
max_wind_speed = df_filled['WindSpeed'].max();
sum_precipitation = df_filled['Precipitation'].sum();
// Print results
print(“Mean Temperature:”+mean_temp);
print("Max Wind Speed:”+max_wind_speed");
print("Total Precipitation:” + sum_precipitation);
df_filled.head();
```

OUTPUT OF SAMPLE1.RC:-

```
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ lex lexer.l
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ gcc lex.yy.c
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ ./a.out
sample1.rc
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ cat output.txt
< | KEYWORD,(input) | >< | CSVFILE,(weather_data.csv) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(df) | >< | OPERATOR,(=) | >< | IDENTIFIER,(read) | >< | PUNCTUATION,(() | >< | PUNCTUATION,(') | >< | CSVFILE,(weather_data.csv) | ><
| PUNCTUATION,(') | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(df_filled) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(miss_value) | >< | PUNCTUATION,(() | >
< | IDENTIFIER,(fill) | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(median) | >< | PUNCTUATION,(() | >< | PU
NCTUATION,()) | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(mean_temp) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df_filled) | >< | PUNCTUATION,([) | >< | PUNCTUATION,(') | >< | IDENTIFIER,(Temperat
ure) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(]) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(mean) | >< | PUNCTUATION,(() | >< | PUNCTUATION,()) | ><
| PUNCTUATION,(;) | >
< | IDENTIFIER,(max_wind_speed) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df_filled) | >< | PUNCTUATION,([) | >< | PUNCTUATION,(') | >< | IDENTIFIER,(Win
dSpeed) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(]) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(max) | >< | PUNCTUATION,(() | >< | PUNCTUATION,()) | >
< | PUNCTUATION,(;) | >
< | IDENTIFIER,(sum_precipitation) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df_filled) | >< | PUNCTUATION,([) | >< | PUNCTUATION,(') | >< | IDENTIFIER,(
Precipitation) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(]) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(sum) | >< | PUNCTUATION,(() | >< | PUNCTUATION,
()) | >< | PUNCTUATION,(;) | >
< | KEYWORD,(print) | >< | PUNCTUATION,(() | >< | DUST,(�) | >< | DUST,(�) | >< | DUST,(�) | >< | IDENTIFIER,(Mean) | >< | IDENTIFIER,(Temperature) |
>< | PUNCTUATION,(:) | >< | DUST,(�) | >< | DUST,(�) | >< | DUST,(�) | >< | OPERATOR,(+) | >< | IDENTIFIER,(mean_temp) | >< | PUNCTUATION,()) | >< | P
UNCTUATION,(;) | >
< | KEYWORD,(print) | >< | PUNCTUATION,(() | >< | STRING_LITERAL,("Max Wind Speed:”+max_wind_speed") | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | KEYWORD,(print) | >< | PUNCTUATION,(() | >< | PUNCTUATION,('"') | >< | IDENTIFIER,(Total) | >< | IDENTIFIER,(Precipitation) | >< | PUNCTUATION,(:)
 | >< | DUST,(�) | >< | DUST,(�) | >< | DUST,(�) | >< | OPERATOR,(+) | >< | IDENTIFIER,(sum_precipitation) | >< | PUNCTUATION,()) | >< | PUNCTUATION,(
;) | >
< | IDENTIFIER,(df_filled) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(head) | >< | PUNCTUATION,(() | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
```

INPUT FOR SAMPLE2.RC:-

```
//reads csv file as input
input weather_data.csv;
// Converts CSV file into data frame
df = read('weather_data.csv');
// Fills missing values with the median
df_filled = df.miss_value(fill,method=ffill);
// Group by Humidity and calculate the mean temperature
grouped = df.groupby('Humidity').agg({'Temperature': 'mean'});
// Reset index to make 'Humidity' a column again
grouped.reset_index();
//creating an output csv file
grouped.to_csv('Output.csv', index=False)
// Creating output csv of the grouped and aggregated data
output Output.csv
```

OUTPUT FOR SAMPLE 2.RC:-

```
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ ./a.out
sample2.rc
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ cat output.txt
< | KEYWORD,(input) | >< | CSVFILE,(weather_data.csv) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(df) | >< | OPERATOR,(=) | >< | IDENTIFIER,(read) | >< | PUNCTUATION,(() | >< | PUNCTUATION,(') | >< | CSVFILE,(weather_data.csv) | ><
| PUNCTUATION,(') | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(df_filled) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(miss_value) | >< | PUNCTUATION,(() | >
< | IDENTIFIER,(fill) | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(method) | >< | OPERATOR,(=) | >< | IDENTIFIER,(ffill) | >< | PUNCTUATION,()) | >< | PU
NCTUATION,(;) | >
< | IDENTIFIER,(grouped) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(groupby) | >< | PUNCTUATION,(() | >< | P
UNCTUATION,(') | >< | IDENTIFIER,(Humidity) | >< | PUNCTUATION,(') | >< | PUNCTUATION,()) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(agg) | >< | PUNCTU
ATION,(() | >< | PUNCTUATION,({) | >< | PUNCTUATION,(') | >< | IDENTIFIER,(Temperature) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(:) | >< | PUNCTUATI
ON,(') | >< | IDENTIFIER,(mean) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(}) | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(grouped) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(reset_index) | >< | PUNCTUATION,(() | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) |
 >
< | IDENTIFIER,(grouped) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(to_csv) | >< | PUNCTUATION,(() | >< | PUNCTUATION,(') | >< | CSVFILE,(Output.csv) |
 >< | PUNCTUATION,(') | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(index) | >< | OPERATOR,(=) | >< | IDENTIFIER,(False) | >< | PUNCTUATION,()) | >< | KEY
WORD,(output) | >< | CSVFILE,(Output.csv) | >surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$
```

INPUT FOR SAMPLE-3.RC:-

```
//reads csv file as input
input weather_data1.csv;
// Converts CSV file into data frame
df=df_filled
//obtained from EXAMPLE CODE 1
df2 =read('weather_data1.csv')
// Merge the two datasets on 'Date'
merged_df = pd.merge(df, df2, on='Date', how='outer',suffixes=('_1', '_2')));
// Interpolate missing values
interpolated_df = merged_df.interpolate(inplace=’False’);
// Print the first few rows of the merged and interpolated data
interpolated_df.head();
```

OUTPUT FOR SAMPLE-3.RC:-

```
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ ./a.out
sample3.rc
surbhi@surbhi-HP-Pavilion-Plus-Laptop-14-eh0xxx:~/Desktop/compilers-2/project$ cat output.txt
< | KEYWORD,(input) | >< | IDENTIFIER,(weather_data1) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(csv) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(df) | >< | OPERATOR,(=) | >< | IDENTIFIER,(df_filled) | >< | IDENTIFIER,(df2) | >< | OPERATOR,(=) | >< | IDENTIFIER,(read) | >< | PUNC
TUATION,(() | >< | PUNCTUATION,(') | >< | IDENTIFIER,(weather_data1) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(csv) | >< | PUNCTUATION,(') | >< | PUNC
TUATION,()) | >< | IDENTIFIER,(merged_df) | >< | OPERATOR,(=) | >< | IDENTIFIER,(pd) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(merge) | >< | PUNCTUATI
ON,(() | >< | IDENTIFIER,(df) | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(df2) | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(on) | >< | OPERATOR,(=) | >< |
PUNCTUATION,(') | >< | IDENTIFIER,(Date) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(how) | >< | OPERATOR,(=) | >< | PUNCTUATION,
(') | >< | IDENTIFIER,(outer) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(,) | >< | IDENTIFIER,(suffixes) | >< | OPERATOR,(=) | >< | PUNCTUATION,(() |
>< | PUNCTUATION,(') | >< | DUST,(_) | >< | INTEGER,(1) | >< | PUNCTUATION,(') | >< | PUNCTUATION,(,) | >< | PUNCTUATION,(') | >< | DUST,(_) | >< | IN
TEGER,(2) | >< | PUNCTUATION,(') | >< | PUNCTUATION,()) | >< | PUNCTUATION,()) | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(interpolated_df) | >< | OPERATOR,(=) | >< | IDENTIFIER,(merged_df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(interpolate) | >< | PUNCT
UATION,(() | >< | IDENTIFIER,(inplace) | >< | OPERATOR,(=) | >< | DUST,(␣) | >< | DUST,(␣) | >< | DUST,(␣) | >< | IDENTIFIER,(False) | >< | DUST,(␣) |
 >< | DUST,(␣) | >< | DUST,(␣) | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;) | >
< | IDENTIFIER,(interpolated_df) | >< | PUNCTUATION,(.) | >< | IDENTIFIER,(head) | >< | PUNCTUATION,(() | >< | PUNCTUATION,()) | >< | PUNCTUATION,(;)
| >
```

PARSER ANALYZER

Parser takes input code is analyzed according to the grammar rules defined using Yacc.The goal of this phase is to check the syntactic structure of the input. Here's a breakdown of its key components:

Operator Precedence and Associativity:

- Operators like +, -, *, /, and % have their precedence defined with %left, and %right for right-associative operators.
- %nonassoc is used to define non-associative operators like <, >, and a special precedence rule LOWER_THAN_ELSE to handle if-else ambiguity.

Token Definitions:

- Tokens like FOR, WHILE, IF, READCSVFUNC, and various operators (ADD_ASSIGN_OPERATOR, EQ_OPERATOR, etc.) are defined to represent keywords, operators, and function calls in the DSL.
- SINGLE_QUOTED_STRING, INTEGER, and IDENTIFIER capture the string literals, integers, and variable names.

Grammar Rules:

- **translation_unit**: The start symbol, representing the entire program. It's composed of one or more declarations.
- **declaration**: Handles type declarations, assignments, function definitions, and input operations.
- **function_call_statement**: Describes function calls like READCSVFUNC(), HEADFUNC(), and others used for CSV manipulation (grouping, merging, etc.).
- **assignment_statement**: Defines how to assign results of function calls to identifiers.
- **control structures**: Implements if-else, while, and for loops, allowing control flow in the DSL.

Parameter and Expression Handling:

- **parameter_list** defines a list of parameters used in functions, while the **expression_list** handles expressions separated by commas.
- The **expression** rule covers various arithmetic expressions and constants.

CSV Functionality:

Functions like READCSVFUNC, HEADFUNC, TOCSVFUNC, and others are tokens for specific operations on CSV files.

- **function_call_statement** details how these functions are invoked with their respective parameters or actions (like MISSVALUEFUNC handling missing values with fill_action).

Error Handling and Cleanup:

- yyerror() reports syntax errors during parsing.
- yywrap() ensures smooth termination when parsing ends.