ProCSV

Language Reference Manual

Team Members:
Tabara Nosiba (tn2341), Project Manager, Tester
Tahiya Chowdhury (tc2672), Language Guru, Tester
Tahsina Saosun (ts2931), System Architect, Tester

1. Introduction

The ProCSV language solves issues pertaining to the manipulation of one or more CSV files. People who regularly analyze, manipulate, and compare data across multiple CSV files for data collection purposes may find it tedious and inefficient in most programming languages. ProCSV will provide users with built-in functions that will make tasks such as cleaning data, comparing, parsing, and visualizing data stored within multiple CSV files. Anyone working with large CSV files will find this language helpful.

Our language is meant to streamline the parsing through CSV files. Since many institutions' data analysis process requires work to parse and visualize insights from traditionally formatted data collection formats, such as CSV. Simplifying this process would improve developers' productivity significantly and would also save companies millions of dollars in the process.

2. Lexical Elements:

a. **Identifiers**:
Identifier refer to csv objects and data types. They consist of one or more REs, lowercase letters, numbers, and underscores.

b. **Reserved keywords**:

int	if	boolean	float	print
for	else if	string	csv	
while	else			

c. **Literals:**

i. **Integer literals**

1. Integer literals are integers with one or more digits that may range anywhere between 0-9

ii. **Float literals**
   1. Float literals are numbers with a fractional value; moreover, they contain a decimal (e.g. The integer 1 as a float would be 1.0)

iii. **String literals**
   1. String literals is a series of one or more characters (e.g. "hello world" is a string literal that consists of the letters as characters as well as a space character)

d. **Operators:**

i. **Arithmetic operators:** +, -, *, /, %

ii. **Logical operators:** &&, ||, !

iii. **Assignment:** =

iv. **Numerical comparators:** ==, <, >, <=, >=

e. **Delimiters:**

i. **Parentheses**
   1. Parentheses are used to contain arguments to be used in function calls

ii. **Semicolon**
   1. Semicolons are used to signify the end of a programmed statement

iii. **Curly braces**
   1. Curly braces are used to contain the scope of a function or control flow

iv. **Comma**
   1. Commas are used to separate arguments or parameters in function calls or declarations

f. **Comments**

ProCSV supports both single line & multi-line commenting. Convention of the single line commenting is a '/* */' symbol that surrounds the comment. The code treats characters within '/* */' as multi-line comments, as well.

Examples:
/* This is a single line comment. */
/*This is the convention for multi-line comments.
ProCSV rocks.*/

3. Data Elements:

i. Primitive data types:

a. Integers:

i. *Declaration*: as type int
ii. *Example*: int y = 5;
iii. *Possible values*: all integers

b. Floats:

i. *Declaration*: as type int
ii. *Example*: float y = 5.5;
iii. *Possible values*: all floating point numbers

c. Booleans:

i. *Declaration*: as type boolean
ii. *Example*: boolean y = true;
iii. *Possible values*: true, false

ii. Non-primitive data type:

a. String:

i. *Declaration*: as type String
ii. *Example*: String s = “hello world”;
iii. *Possible values*: any string literal
iv. *Functions*: string.equals(string s) → checks if two strings are equal. Returns true if they are equal and false if they are not.

b. csv: csv stores data parsed from the CSV file
i. *Declaration:* as type csv
Ii. *Example:* csv input_csv1 = read(‘sample1.csv’);

4. Functions

i. **Built-in functions:** These following functions are provided and built into the ProCSV language.

read_csv()	Reads in a CSV file
parse()	Parses data from the CSV file based on the specified delimiter
print()	Prints all data types
tablify()	Formats the data the user wants to put into a table format

ii. **User-defined functions:**

a. Programmers may define their own functions in the following format:

```
return_type function_name(parameter 1, ...) {
        logic statements;
        .
        .
}
```

b. Programmers may then call the functions like such:

```
function_name(argument 1, ...);
```

iii. **Library functions:** The following are functions that are supported in a library within ProCSV. Users can import and use these functions in their own code.

merge()	Merges two different CSV files into one
find()	Looks for and returns a specific piece of data based on the argument
sim()	Compares data inside two different

	CSV files and returns a csv data type of common data

5. Control Flow Statements

i. if-else/else if/else statements: Logical conditioning statements that help users separate their code in specific blocks.

Example:

```
if (boolean expression){
        do something here;
}
else if (boolean expression){
        do something here;
}
else{
        do something here;
}
```

ii. for loops: Logical conditioning statement to help users execute a specific block of code multiple times.

Example:

```
for (condition of looping statement){
        do something here;
}
```

iii. while loops: Logical conditioning statement to help users execute a specific block of code until a condition no longer applies.

Example:

```
while (condition of statement){
        do something here;
}
```

6. **Sample Programs**:

```
/*
```

```
Takes in two csv files as arguments and returns common data
*/

csv findSimilarData(csv c1, csv c2){
        csv similarData = sim(c1, c1);
        return similarData;
}

//Calling the function inside the main method
csv main(String[] args){
        csv input_csv1 = read(‘sample1.csv’);
        csv input_csv2 = read(‘sample2.csv’);

        csv result_csv = findSimilarData(input_csv1, input_csv2);
}


/*
Takes in two csv files as arguments, merges them into one csv file, and displays
data in a table
*/

table createTable(csv c1, csv c2){
        csv merged_csv = merge(c1, c2);
        table dataSet = tablify(merged_csv);
}

//Calling the function inside the main method
int main(String[] args){

        csv input_csv1 = read(‘sample1.csv’);
        csv input_csv2 = read(‘sample2.csv’);
```

```
    table result_table = createTable(input_csv1, input_csv2);
}
```