Tabulate
Language Specification

Anshul Sangrame
CS21BTECH11004

Varun Gupta
CS21BTECH11060

Gautam Singh
CS21BTECH11018

Contents

***Abstract*—This document provides the language specification for *Tabulate*, a domain specific language (DSL) that provides programming constructs to automate spreadsheet processing efficiently.**

1. Introduction

1.1. Motivation

Spreadsheets are an integral part of our lives. Whether it comes to creating timetables, bookkeeping possessions or tabulating marks, it is difficult to imagine life without spreadsheets. Unfortunately, most spreadsheet softwares like Microsoft Excel and Google Sheets are *What You See Is What You Get* (WYSIWYG) editors. These softwares do not offer a very good programming interface, which in most cases can automate jobs much faster.

That's where Tabulate comes in. With high level programming constructs to abstract the implementation of seemingly complex operations, Tabulate makes it possible to *program* your spreadsheet.

1.2. Goals

Tabulate aims to be the go-to DSL for those who manage many spreadsheets in their everyday life. It aims to automate the tedious process of repetitive entries, updates, and formulae with high performance and efficiency. Built on top of C++, Tabulate aims to provide the programmer with more control over their spreadsheet.

2. Lexical Conventions

2.1. Comments

Tabulate has only one kind of comments; these are of the form `#...#`. Notice that this style of comments can be either single line or multiline.

2.2. Whitespaces

Whitespaces in Tabulate are useful only in separating tokens. Excess whitespaces are ignored.

2.3. Reserved Keywords

Some of the reserved keywords in Tabulate are shown in Table 1.

IF	PI	INT	AND	RETURNS	SUM
ELSE	TRUE	TABLE	OR	INF	PROD
IMPORT	FALSE	DOUBLE	NOT	BREAK	NROWS
WHILE	CELL	STRING	TYPEOF	CONTINUE	NCOLS
FUN	RANGE	CLASS	RETURN	MAIN	NCELLS
VOID	GET	FORMULA			

TABLE 1. KEYWORDS IN TABULATE.

2.4. Identifiers

Identifiers in Tabulate can contain characters, digits and underscore. However, they must start with either a character or underscore. Identifiers are case-sensitive.

2.5. Punctuators

The list of punctuators in Tabulate along with their meanings is given in Table 2.

Punctuator	Description
;	Statement terminator
,	Array separator
.	Member access
:	Range specifier
{	Block start
}	Block end
(	Open parenthesis
)	Close parenthesis
[	Open bracket (for arrays, etc.)
]	Close bracket
”	String delimiter

TABLE 2. PUNCTUATORS IN TABULATE.

3. Datatypes

Datatypes in Tabulate are of two types as shown below.

3.1. Primitive Datatypes

The list of primitive datatypes offered by Tabulate are shown in Table 3.

3.2. Non-primitive Datatypes

The list of non-primitive datatypes offered by Tabulate are shown in Table 4.

4. Operators

The list of operators provided by Tabulate are shown in Table 5.

Datatype	Description	Example
int	64-bit signed integer value	int x = 1;
double	64-bit signed numerical value, including decimal values	double x = 0.05;
string	sequence of characters	string name = ”hastar”;
bool	represents boolean values either 0/1 or true/false	bool flag = true;
date	represent dates	date today = 2023-04-15;
time	represents time	time now = 15:30:45;

TABLE 3. PRIMITIVES IN TABULATE.

Datatype	Description	Example
cell	Single cell in the sheet	cell A1 = 5;
range	Range of cells	range data = A1:A10;
array	List of values of same primitive data type	int[5] num = [1,2,3,4,5];
table	Represents structured range of cells with headers and data rows	table sample = A1:D100;
formula	Datatype that holds formula	formula f1 = SUM(A1:A10) / 10;
struct	User-defined datatype combining multiple primitive and/or non-primitive data types	class example { int id; string name; double price; }; class example = {id: 101, name: ”widget”, price: 19.99}

TABLE 4. NON-PRIMITIVES IN TABULATE.

5. Statements

5.1. Simple Statements

In Tabulate, the simplest type of statements are *simple statements*. These statements are delimited with a semicolon. Expression statements can be of the following types.

5.1.1. Declaration Statements. The syntax for declaration statements is given in Listing 1. Note that Tabulate allows for multiple declarations in a single statement for the same data type, with the help of comma for separation.

```
# Declarations in Tabulate #
int a, b; # Multiple declarations #
bool fl; # Single declaration #
```

Listing 1. Declaration Statements in Tabulate

5.1.2. Assignment Statements. The syntax for assignment statements is given in Listing 2. Note that constants can appear only in the right hand side of assignment statements in Tabulate.

```
# Assignment statements in Tabluate #
a = 3;
```

Category	Operator	Description	Associativity	Valid Operands	Example
Arithmetic	+	Addition	Left to Right	cell, int, double	A1 + A2
	-	Subtraction			B1 - 5
	*	Multiplication			A1 * 10
	/	Division			C1 / 2
	%	Modulus			A1 % 2
	^	Exponent			C1 ^ 2
Comparison	==	Equals	Left to Right	cell, bool	A1 = A2
	!=	Not Equals			B1 != B2
	>	Greater Than		cell, int, double	A1 > 100
	<	Less Than			A2 < 50
	>=	Greater Than or Equal to			B1 >= 25
	<=	Less Than or Equal to			B2 <= 75
Logical	AND	Logical AND	Left to Right	cell, bool	(A1 > 100) AND (A2 < 50)
	OR	Logical OR			(B1 = 25) OR (B2 != 75)
	NOT	Negation			NOT (A1 = A2)
Assignment	=	Assign	Right to Left	cell	A1 = 5
Unary	-	Unary Minus	Right to Left	cell, int, double	-A1
	TYPEOF	Type determination		any	TYPEOF(F6)
	~	Logical NOT		bool	~C3
Reference	:	Cell Range Reference	None	cell	A1:A10
	!	Table Reference		table	Table2!A1
Access	.	Class Member Access	Left to Right	class	myClass.name

TABLE 5. OPERATORS IN TABULATE.

```
# We can declare and assign too. #
string s = "string";
```

Listing 2. Assignment Statements in Tabulate

5.1.3. Function Calls. The syntax for function calls is given in Listing 3. Note that nested function calls are allowed in Tabulate.

```
# Function calls in Tabulate #
c = ADD(a,b);
# Nested function calls #
d = ADD(MUL(a,b),c);
```

Listing 3. Function Calls in Tabulate

5.1.4. Return Statements. The syntax for return statements is given in Listing 4.

```
# Return statements in Tabulate #
return x;

# One can simply return, meaning
the same as return void #
return;
```

Listing 4. Return Statements in Tabulate

5.1.5. Expression Statements. Expression statements in Tabulate must contain a left and right hand side, as shown in Listing 5.

```
int a = 5 + 13; # evaluates to 18 #
double e = 20.0 * 0.25; # evaluates to 5.0 #
double g = 10.8 / 2; # evaluates to 5.4 #
int h = 13 % 3; # evaluates to 1 #
double j = 2 ^ 3; # exponentiation, evaluates to 8 #

bool l = (10.2 > 4); # evaluates to TRUE #
bool p = (5 < 3 OR 4 > 2); # evaluates to TRUE #
bool q = NOT(5 == 6); # evaluates to TRUE #

int a = 5;
int b = −a; # Unary negation, evaluates to −5 #

# Assuming a function TYPEOF() that returns a string
    with the type
string e_type = TYPEOF(5); # evaluates to "int" #
string f_type = TYPEOF(10.5); # evaluates to "double"
    #

# Assuming cells from A1 to A5 have values 1, 2, 3, 4,
    5 respectively. #
Range myRange = A1:A5; # References cells from A1
    to A5 #

double sumRange = SUM(myRange); # Using a SUM
    function, evaluates to 15 #

# Assuming a table named "SalesData" exists in the
    sheet. #
Table myTable = SalesData;

int rowCount = ROWS(myTable); # Returns the number
    of rows in "SalesData" #
int colCount = COLUMNS(myTable); # Returns the
    number of columns in "SalesData" #
```

```

# Assuming a class "Student" with user−defined "name" and "grade". #
Student s1 = {name: "hastar", grade: "A"}

string studentName = s1.name; # Accesses the name of the student, evaluates to "John" #
string studentGrade = s1.grade; # Accesses the grade of the student, evaluates to "A" #
```

Listing 5. Expression Statements in Tabulate

5.1.6. Jump Statements. Tabulate supports the jump statements `break` and `continue`, as shown in Listing 6.

```
fun firstPositive(range: Range) returns double {
    double positiveVal = −INF;
    int rangeLength = LENGTH(range);
    int currIdx = 0;

    while(currIdx < rangeLength) {
        double currentVal = GET(range, currIdx);

        if(currentVal > 0) {
            positiveVal = currentVal;
            break; # Exit the loop as we found a positive value #
        }

        if(currentVal == 0) {
            currIdx = currIdx + 1;
            continue; # Skip this iteration and move to the next value #
        }

        currIdx = currIdx + 1;
    }

    return positiveVal;
}
```

Listing 6. Jump Statements in Tabulate

5.2. Compound Statements

In their simplest form, compound statements in Tabulate contain one or more expression statements nested within scope braces. However, compound statements can be nested in other compound statements. An example is shown in Listing 7.

```
int val1 = GET(A1);
int val2 = GET(A2);
double res;

if(val1 > 50) {
    if(val2 > 20) {
        res = val1 ∗ 0.8 + val2;
    } else {
        res = val1 ∗ 0.9;
    }
} else {
    if(val2 < 10) {
        res = val1 + val2 ∗ 1.2;
    } else {
        res = val1 + val2;
    }
}
```

Listing 7. Compound Statements in Tabulate

5.3. Selection Statements

Tabulate also offers a construct for selecting statements to be executed based on one or more conditions. The syntax for such selection statements is illustrated in Listing 8

```
# Selection statements in Tabulate #
bool fl1, fl2;

#...#

if (fl1) {
    # CODE #
} else if (fl2) {
    # MORE CODE #
} else {
    # SOME MORE CODE #
}
```

Listing 8. Selection Statements in Tabulate

5.4. Iteration Statements

Tabulate offers one iteration statement construct, illustrated in Listing 9.

```
# Iteration statements in Tabulate #
bool fl;

#...#

while (fl) {
    # CODE #
}
```

Listing 9. Iteration Statements in Tabulate

6. Functions

6.1. Definition

The syntax for definition of a function is given in Listing 10.

```
fun addNumbers(a: int, b: int) returns int {
    int result = a + b;
```

```
    return result;
}

fun findMax(range: Range) returns double {
    double maxVal = −INF;
    int rangeLength = LENGTH(range);
    int currIdx = 0;

    while(currIdx < rangeLength) {
        if(GET(range, currIdx) > maxVal) {
            maxVal = GET(range, currIdx);
        }
        currIdx = currIdx + 1;
    }

    return maxVal;
}
```

Listing 10. Function Definitions in Tabulate

6.2. Main Function

Programs that can be executed must contain a `main` function, as illustrated in Listing 11.

```
import ”calculations.tblt”;

fun main() returns void {
    # ... your main code here ...
    return;
}
```

Listing 11. Usage of Main Function in Tabulate

7. Importing Code

Tabulate offers a powerful feature in the form of importing source code for use in other codes through the `import` directive. The following points regarding its use should be noted.

1) `import` directives occur at the top of the source code.
2) Source codes containing a `main` function cannot be imported in other codes.
3) Imports should not be circular, else the code will not compile.

An example of the use of `import` is in Listing 11.

8. Standard Library Functions

Tabulate contains an extensive standard library to handle your basic spreadsheet requirements without having to import anything. These requirements are implemented purely using standard C++ libraries.