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	GET	INT	AND	RETURNS	SUM
ELSE	TRUE	TABLE	OR	INF	PROD
VOID	FALSE	DOUBLE	NOT	BREAK	MUL
WHILE	CELL	STRING	TYPEOF	CONTINUE	DIV
FUN	RANGE	CLASS	RETURN	MAIN	MOD
FORMULA	POW	FLOOR	CEIL	BAND	BXOR
BOR	BNOT	BRS	BLS		

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	Name
;	Semicolon
,	Comma
.	Dot
:	Colon
{	Left curly bracket
}	Right curly bracket
(	Left parenthesis
)	Right parenthesis
[	Left square bracket
]	Right square 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 = (1:5:1, 1:5:1); range rng = (1:5:1); range col = (2, 1:5:1);
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 tab1;
formula	Datatype that holds formula	formula f1 = SUM(1,1:10:2) / 10;
class	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. The following points are important for declaration statements in Tabulate.

1) The declaration modifier `let` is used for variables whose values can change later on in the program.
2) The declaration modifier `const` is used for variables whose values will not change in the program. Notice that `const` can be used *only* with assignments.
3) Tabulate supports **dynamic typing**. Variables can take different types after their declaration, based on the assigned value.
4) Tabulate supports single and multiple declarations.

Category	Operator	Description	Associativity	Valid Operands	Example
Arithmetic	ADD	Addition	Left to Right	cell, int, double	ADD(2,3)
	SUB	Subtraction			SUB(myTable[1,2],5)
	MUL	Multiplication			MUL(myTable[1:5:1],10)
	DIV	Division			DIV(myTable[1],myTable[2])
	MOD	Modulus			MOD(12,5)
	POW	Exponent			POW(myTable[3],4)
Comparison	==	Equals	Left to Right	cell, bool	cell1 = cell2
	!=	Not Equals			cond1 != cond2
	>	Greater Than		cell, int, double	cell1 > 100
	<	Less Than			cell2 < 50
	>=	Greater Than or Equal to			cell1 >= 25
	<=	Less Than or Equal to			cell2 <= 75
Logical	AND	Logical AND	Left to Right	cell, bool	(cell1 > 100) AND (cell2 < 50)
	OR	Logical OR			cond1 OR cond2
	NOT	Logical NOT			NOT(cond)
Bitwise	BOR	Bitwise OR	Left to Right	cell, int, double, bool	BOR(cell1, 2)
	BAND	Bitwise AND			BAND(cell1, 2)
	BXOR	Bitwise XOR			BXOR(cell1, 2)
	BNOT	Bitwise NOT			BNOT(cell1, 2)
	BLS	Bitwise Left Shift			BLS(cell1, 2)
	BRS	Bitwise Right Shift			BRS(cell1, 2)
Assignment	=	Assign	Right to Left	cell	cell1 = 5
Unary	-	Unary Minus	Right to Left	cell, int, double	-cell1
	TYPEOF	Type determination		any	TYPEOF(cell6)
	~	Logical NOT		bool	~cell3
Reference	:	Range Reference	None	cell	1:5:1
	[ ]	Table Reference		table	myTable[1:5:1,1:5:1]
Access	.	Class Member Access	Left to Right	class	myClass.name

TABLE 5. OPERATORS IN TABULATE.

```
# Declarations in Tabulate #
let x, y, z; # Multiple declarations #
let c = ”string”; # Single declaration #
```

Listing 1. Declaration Statements in Tabulate

5.1.2. Assignment Statements. The syntax for assignment statements is given in Listing 2. Either expressions or constants can appear on the right hand side of the assignment statement. Multiple assignments are also possible as shown.

```
# Assignment statements in Tabluate #
a = 3;
# We can declare and assign too #
let s = ”string”, s1 = ”string1”, c1 = 20;
# const modifiers require assignment #
const b = 2, b1 = 21.12, b2 = ”22”;
```

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.

```
# Expressions in Tabulate #
let a = ADD(5, cellA); # evaluates to 18 #
let j = POW(2, 3); # 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 #

let a = −5; # Unary negation #

let t1 = TYPEOF(5); # evaluates to "int" #
let t2 = TYPEOF(10.5); # evaluates to "double" #

let myRange = 1:5:1;

let sumRange = SUM(myTable[myRange]); # Using a
      SUM function, evaluates to 15 #

# Assuming there exists a table called SalesData #
let myTable = SalesData;

# Assuming a struct Student with members name and
      grade #
let s1 = {name: "hastar", grade: "A"}
let studentName = s1.name; # Accesses the name of the
       student, evaluates to "hastar" #
let 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.

```
# Jump statements in Tabulate #
fun firstPositive(range: Range) returns double {
      let positiveVal = −INF;
      let rangeLength = LENGTH(range);
      let currIdx = 0;
      while(currIdx < rangeLength) {
            let currentVal = GET(range, currIdx);
            if(currentVal > 0) {
                  positiveVal = currentVal;
                  break; # Exit the loop #
            }
            if(currentVal == 0) {
                  currIdx = currIdx + 1;
                  continue; # Skip this iteration #
            }
            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.

```
# Compound statements in Tabulate #
let val1 = GET(cell1);
let val2 = GET(cell2);
let 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 #
let 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 #
let fl = TRUE;

#...#
```

```
5
6  while (fl) {
7      # CODE #
8  }
```

Listing 9. Iteration Statements in Tabulate

6. Functions

6.1. Definition

The syntax for definition of a function is given in Listing 10. Each function *must* contain a return statement.

```
1  # Functions in Tabulate #
2  fun addNumbers(a, b) {
3      let result = a + b;
4      return result;
5  }
6
7  fun findMax(range) {
8      let maxVal = -INF;
9      let rangeLength = LENGTH(range);
10     let currIdx = 0;
11
12     while(currIdx < rangeLength) {
13         if(GET(range, currIdx) > maxVal) {
14             maxVal = GET(range, currIdx);
15         }
16         currIdx = currIdx + 1;
17     }
18
19     return maxVal;
20 }
```

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.

```
1  # Main function in Tabulate #
2  fun main() {
3      # ... your main code here ... #
4      return;
5  }
```

Listing 11. Usage of Main Function in Tabulate

7. 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.