ChartLan

A Language for Chart

COMS W4115 PLT project proposal
Professor: Stephen A. Edwards

Yibo Zhu	**(yz2486@columbia.edu)**
Xiuming Dou	**(xd2138@columbia.edu)**
Xiao Xu	**(xx2165@columbia.edu)**
Ziyue Chen	**(zc2239@columbia.edu)**
Xiang Ma	**(xm2151@columbia.edu)**

1. Introduction

ChartLan is a language specified in table creating, information storing, retrieving and modifying. When programming in ChartLan, users can easily create a Table and store various data type from integers, floats to strings and Booleans in it. ChartLan also provides easy ways to retrieve or modify stored entries in a table. Operations like addition and concatenations between data storages like arrays and tables are designed to be simple and easy in ChartLan. There are many built-in functions to help users to process the data stored in the table. Users can also draw different kind of diagrams to illustrate the data in the table.

2. Motivation

Today, we live in a society composed of large quantities of data. Therefore, data processing is extremely important in everyday life. Software like Excel, MATLAB may have many professional statistical functions. However, our programming language, Chartlan, is designed for users to creating table and processing data by typing a few lines of code. With the help of new data type and new defined operators, users can simplify the process of creating table, inserting and deleting data. It also includes many fundamental and useful functions that meet almost every assignment of daily work.

3. Lexical conventions

(1)Comments

Single line comments

The characters /* introduce a comment, which terminates with the characters */.

Multiple line comments

The characters #* introduce multiple line comment, which terminates with the characters *#

(2)Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. Upper and lower case letters are considered different.

(3)Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

Int	integer number
Float	float number
String	serial of characters

Array	list of integers, floats or strings
Table	list of array integers, floats or strings
Boolean	boolean
for	loop control
If…else…	conditional control
Def	function definition

(4)Constants

Boolean constants	true, false
Integer constants	…-2 -1 0 1 2…
Floating constants	integerpart . fraction part
String constants	“abcedf “ or “chart\”lan “
Array constants	(some constants of integer or float or string separated by “,”) e.g. (1,2,3,”lol”,3.22)
Table constants	((row1) (row2) (row3)…) where row1, row2, row3… are array constants separated by parentheses, e.g. ((1,2,3)(“a”,”b”,”c”)(1.2,3.4,5.6))

4. Declarations

General rule: Type-specifier identifier = expression;

For types integer, string , float, identifier = value of the identifier;
Eg: int x = 3; string c = “this is a string”, float f = 3.1415

As for the declaration of array variable:
Array identifier = (integers or floats or strings separated by “,”)

As for the declaration of table variable:
Table identifier = (array1 array2 …)

5. Operators

Addition operators:
The addition operator + and group left to right

Expression + *expression*

The binary + operator indicates addition.
Integer + integer gives integer addition value.
Float + float gives back float addition value.
String + string gives the concatenation of the two strings

Addition between integer and float treats the integer as a float whose fraction part is 0 and returns float value.

When two arrays perform an addition, each of the array elements will be added up in pairs. Those elements that do not have a pair to match will be appended in the back of the result array.

Eg. (1, 2, “ha”, 3) + (1, 1,”ha”) gives back (2, 3,”haha”, 3)

Finally, addition between Tables will pair wise add up the corresponding rows of the two tables and append those rows that do not have a matching pair in the end of the table.

Eg: Table1 is ((1, 2, 3) (4, 5, 6)), Table2 is ((7 ,8 ,9)) then, Table1 + Table2 will be a new Table with 2 rows: ((8,10 ,12) (4 ,5, 6)).

Subtraction operator:

The addition operator - and group left to right, and it is like the inverse operation of addition.

Expression - expression

Perform normal subtractions on two expressions.

The binary - operator indicates subtraction.

Integer - integer gives integer subtraction value.

Float - float gives back float subtraction value.

String1 – string2 gives remained string that deletes the first matching string2 from string1’s substring.

Subtraction between integer and float treats the integer as a float whose fraction part is 0 and returns float value.

When two arrays perform a subtraction, each of the array elements will subtract in pairs, those elements that do not have a pair to match will be appended in the back of the result array.

Eg. (1, 2, “haha”, 3) - (1, 1,”ha”) gives back (0, 1,”ha”, 3)

Finally, subtraction between Tables will pair wise subtract the corresponding rows of the two tables and append those rows that do not have a matching pair in the end of the table.

Eg: Table1 is ((1, 2, 3) (4, 5, 6)), Table2 is ((7 ,8 ,9)) then, Table1 - Table2 will be a new Table with 2 rows: ((-6,-6 ,-6) (4 ,5, 6)).

Multiplication operator:

The multiplication operators * group left to right

*Expression * expression*

The binary * operator indicates multiplication.

Integer * integer gives integer multiplication value.

Float * float gives back float multiplication value.

Multiplication between integer and float treats the integer as a float whose fraction part is 0 and returns float value.

Note when an integer or a string or a float multiplies with an Array will insert the item into the array. 1 * (2, 3) will insert the integer 1 at the head of the array. Thus give back (1, 2, 3); (2, 3)*1 will put integer 1 at the end of the array giving back (2, 3, 1).

When two arrays perform a multiplication, the second array will be appended to the first. eg: (1 ,2, 3) * (4, 5 ,6) gives back (1 ,2 ,3 ,4, 5 ,6)

Finally, Multiplications between Tables will append the second table's rows onto the end of the first's provided: Table1 is ((1, 2, 3) (4, 5, 6)), Table2 is ((7 ,8 ,9) (10,11,12)) then, Table1 * Table2 will be a new Table with 3 rows: ((1 ,2 ,3) (4 ,5, 6) (7, 8, 9) (10,11,12)).

Division operator:

The division operators / group left to right, the division is like the inverse operation of multiplication.

Expression/expression

The binary / operator indicates division.

The integer, float data type can perform division as commonly use, and divisor cannot be zero.

Integer/integer gives back integer value. If the actual result is not an integer, it will return the integer part of the result.

Float/float gives back float value.

Division between integer and float treats the integer as a float and returns float value.

When an array divides an integer or a string or a float, the array will delete the first matching item from the array, e.g: (2,3,1)/1=(2,3), (3,4,4,5)/4=(3,4,5). (4,5,6)/2 will return the original array (4,5,6).

The division between table and array will delete the array from the table and return the remaining table. E.g.((1,2,3)(3,4,5)(4,5,6))/(4,5,6)=((1,2,3)(3,4,5)). If the array is not contained in the table, the original table will remain the same.

The division between two arrays and two tables is not allowed, e.g.(1,2,3,4)/(1,4)will throw an error, which can be replaced by (1,2,3,4)/1/4 or other function to achieve the same result.

Arithmetic operator:

Expression .+ expression
Expression .- expression
Expression .*expression
Expression ./ expression

The arithmetic operators are used when users want to do the operation to the every element in array or table.

E.g.: Array (2,3,1).+3= (5,6,4);

Array (2,3,4).-1=(1,2,3);

E.g.: Table ((1,2,3) (4,5,6)).*3=((3,6,9)(12,15,18));

Table ((1.0, 2.0, 3.0) (4.0, 5.0 ,6.0)) ./2=((0.5,1.0,1.5)(2.0,2.5,3.0));

Equality operator:

Expression == expression

returns Boolean constants true if the two expressions are identical and false otherwise.

Expression != *expression*

returns Boolean constants false if the two expressions are identical and true otherwise.

Expression1 >(>=) expression2

returns Boolean constants true if expression1 is greater (not less than) expression2 and false otherwise.

Expression <(<=) expression

returns Boolean constants true if expression 1 is less (not greater than) expression2 and false otherwise.

Assignment operator:

Identifier = expression.

Assigns an expression's value to the identifier.

Logical operator:

Expression && expression

returns Boolean constants true if the two expressions are both true and false otherwise.

Expression || expression

returns Boolean constants true if the one of the two expressions is true and false otherwise.

6. Statements

Expression statement:
Most statements are expression statements, which have the form *expression* ;

Conditional statement:
The two forms of the conditional statement are
if (*expression*) {*statement}* else {*statement}*
In both cases the expression is evaluated and if it is nonzero, the first substatement is executed. In the second case, the second substatement is executed if the expression is 0.

While statement:
The while statement has the form
while (*expression*) *statement*
The statement is executed repeatedly so long as the value of the expression remains nonzero.
The test takes place before each execution of the statement.

Function definitions

Def typespecifier functionname (typespecifer identifier1, typespecifer identifier2 …) { function body}

Where typespecifier indicates the return type of the function which could be either int, float,string,array or table.

Arguments consists of arguments of the function which are identifiers with their type specified in the front.

Def is the keyword that specifies a function definition.

Functionname is a combination of any letters or integers but the first must be a letter.

Eg: *Def int max(int a, int b){ /*gives back the bigger one of the two integers*/*

```
If ( a > b) {
    Return a;
}
Else{
    Return b
}
```

Some of built-in functions:
Int fread(String filename)
Int fwrite(String filename, Table t)
Table createTable()
Array getRow(Table t, int index)
Array SizeofTable(Table t)
Int Length(Array a)

Table insertrow(Table t, int index)
Table deletetrow(Table t, int index)

7. Sample program

This is a very simple program of building and keeping track of person's phone numbers.

We want a table which first column is the name of the contacts and the second column contains the corresponding phone number of the contact.

```
Table contactinfo =
CreateTable( ("Peter","212-505-3333")("Jane","374-864-5123")("John","974-505-2222"))

/* we can store our table in a file */
Fwrite("mycontactinfo.txt", contactinfo);

/* and when we need it, we can read the table from file */
contactinfo = fread("mycontactinfo.txt")

/*now say we want to add another contact */
Array SmithEntry = ("Smith","777-223-2222");
Contactinfo = contactinfo * SmithEntry;    /*append the smith's contact into our table

/* we can define functions to retrieve data from the contactinfo table */
Def    String    SearchPhoneNumber( Table t, String name){
      Int i = SizeOfTable(t)[0];    /*built in function SizeOfTable returns array whose first element is the number of rows table t contains and the second the number of columns. Calling SizeofTable(t) with indexing[0] to get number of rows t contains.
      While ( i > = 0){
            If (getTableRow(t,i)[0] == name){ //getTableRow takes a table t, an integer i and returns the ith row of the table in an array.
                  Return getTableRow(t,i)[1];
             }
            i = i – 1;
      }
}
/* now we can use the function defined to look up contact information*/
String Johnsphonenumber= SearchPhoneNumber(contactinfo, "John");
/* which will give back John's phone number "974-505-2222" */
```