Calcul^2 Programming Language Reference Manual

Junde Huang

Kewei Ge

Zhan Shu

Jinxi Zhao

Wenting Yin

October 28, 2013

CONTENTS

1 Introduction

1.1 Motivation

CalCul^2 is a programming language focusing on math calculation such as calculus. Like a powerful calculator, CalCul^2 can calculate values, deal with math functions, and even do calculus. It is very useful and convenient.
Although the existing languages, such as C or Java, could also handle math computations, they often requires complicated programming, which is very tedious for users. Our CalCul^2 will allow users to describe and manipulate calculus in a much simpler way.

1.2 Language Description

CalCul^2 is a simple yet powerful programming language that deals with limits and the differentiation and integration of functions of one or more variables. CalCul^2 supports basic mathematical as well as calculus operations, which involves ordinary differential equation and definite integral calculation. Instead of redundancy types, developers can solve problems with clear and simple codes in CalCul^2. E.g. To make codes concise, CalCul^2 does not require a type definition for each variable, even a function, instead it will automatically recognize the type and might adjust it with the processing of programs. Furthermore, CalCul^2 also provides ability to simple evaluation of functions. CalCul^2 is a concise language for calculus: just using less to accomplish more.

2. Program Definition

CalCul^2 is a programming language that deals with limits and the differentiation and integration of functions of one or more variables.

3. Lexical Conventions

3.1 Comments

In line comments are represented by ##, while block comments are delimited by #* and *#.

3.2 Identifiers

Identifiers are comprised of uppercase letters, lower letters, digits and underscore _. The identifiers are case sensitive. The first character cannot be a digit, nor can it be an underscore_.

3.3 Keywords

The following keywords are reserved:

int	Pi
float	e
bool	sqrt
vector	sin
function	cos
if	tan
else	asin
for	acos
while	atan
log	ln

3.4 Constants

3.4.1 *Integer constants*
Integers are represented by a combination of digits and nothing else.
Only decimal representations are allowed. Example:

```
## Define an integer
x=2;
```

3.4.2 *Float constants*
Floats must have exactly one decimal point, with digits on either side.
Only decimal representations are allowed. Example:

```
## Define floats
a=3.0;
b=0.;
c=.155;
```

3.4.3 *Boolean constants*
Booleans can be either ***true*** or ***false*** (case sensitive).

3.5 Operators

3.5.1 *Arithmetic Operators*

Operator	Definition
+	Addition
-	Subtraction
*	Multiplication
/	Division

^	Power
//	Integer division
%	Module
=	Assignment
’	Derivative
@	Integral

3.5.2 *Logic Operators*

<	less than
>	greater than
<=	equal to or less than
>=	equal to or greater than
==	equal to
!=	not equal to
&&	and
||	or
□	not

3.6 Punctuators

Symbol	Symbol Definition
;	Marks the end of a statement
[ ]	Marks a vector
( )	Used for grouping parameters in function call, or indicates priority in expression
{}	Marks the beginning and end of a group of statements
:	Used to get output

,	Separates parameters of a function

4. Data Types

Calculus Calculator Language don't need the user to designate data type for every variable. It will be determined by the initial definition. The build-in data types are as follows:

Type	Definition
int	An Integer value
float	The Double type is used for all floating-point calculations
bool	The Boolean type can take of one of two possible values –true or false
vector	A vector is a contiguous region of storage for any zero or more of any given type in Culcus^2.
function	A function is a mathematical function with several variables

5. Expression and Operations

5.1 Primary expressions

5.1.1 *identifier*
Its type can be a variable or a function without explicit declaration.

5.1.2 *constant*
Its type is int or float.

5.1.3 *(expression)*

5.1.4 *identifier[expression]*
It yields the value at the index of a vector.

5.2 Unary operators

5.2.1 *!expression*
Performs negation on the expression, which must be boolean.

5.2.2 *expression'*
Performs derivation of function with respect to a variable, where there must be only one variable in this function.

5.3 Basic Arithmetic Operators

As a calculating language we definitely need to bring users the basic arithmetic operators, including '+', '-', '*', '/', '^'(power), '//'(integer division), '%'(module). And also we provide logical operators like '<', '>', '<=', '>=', '==', '!=', '&&', '||' and '!'.

5.3.1 Multiplicative operators

5.3.1.1 *expression * expression*
Multiplication operator * is valid between two integers, two floats, or an integer and a float.

5.3.1.2 expression / expression
Division operator / is valid between two floats or an integer and a float.

5.3.1.3 *expression // expression*
Integer division operator // is valid between two integers and yields a value of integer.

5.3.1.4 *expression % expression*
Mod operator % is valid between two integers to give the remainder of two expressions.

5.3.1.5 *expression1 ^ expression2*
Power operator ^ is used to calculate the value of base expression1 with power of expression2.

5.3.2 Additive operators

5.3.2.1 *expression + expression*
The + operator returns the sum of two expressions.

5.3.2.2 *expression - expression*
The - operator returns the difference between two expressions.

5.3.3 Comparison operators

5.3.3.1 *expression == expression*
Returns a boolean value whether these two expressions are equal.

5.3.3.2 *expression > expression (expression >= expression)*
Returns a boolean value whether the left expression is larger (larger or equal) than the right expression.

5.3.2.2 *expression < expression (expression <= expression)*

Returns a boolean value whether the left expression is smaller (smaller or equal) than the right expression.

5.3.2.2 *expression != expression*
Returns a boolean value whether these two expressions are not equal.

5.3.4 Logical operators

5.3.4.1 *expression && expression*
Logical AND between two boolean expressions.

5.3.4.2 *expression || expression*
Logical OR between two boolean expressions.

5.4 Advanced Arithmetic Operators

We will provide advanced arithmetic operators as follows:

5.4.1 *f(expression1, expression2, ...)*
Assume f is a function with respect to multiple variables or a single variable, thus f(expr1, expr2,..) operator is used to get the value of this function when these variables are specified a value. The return type is a value (integer or float). E.g. f(x, y) = 3 * x ^ 2 + y // 2; Hence we get f(2, 3) = 13.

5.4.2 *f'(expression)*
Unlike the unary operator *f'* mentioned above, where the function *f* has only a single variable, here the derivation operator supports multiple variables. That is, it gives the derivation of function *f* with respect to some *expression*. The return type is a function. E.g. f(x, y) = 3 * x ^ 2 + 3 * x * y; We have f'(x) = 6 * x + 3 * y.

5.4.3 *f'(expression1, expression2, ...)*
This operator gives the gradient of function *f* at *expression1, expression2* and possibly other variables, which will be returned in type of vector.

5.4.4 *f@expression1(expression2, expression3)*
Definite integral operator @ is used for calculate the definite integral of function *f* with respect to *expression* in the range from *expression2* to *expression3*. The return type is a value (integer or float). E.g. f(x) = 3 * x ^ 2 + x; We have f@x(3, 10) = 1018.5.

All these Arithmetic Operators except *f(expr1, expr2,...)* are only allowed to be applied on pure mathematical functions(i.e. function not include recursions, loops or condition controls).

5.5 Other Arithmetic Values

5.5.1 *Pi*

The number π (*Pi*) is a mathematical constant that is the ratio of a circle's circumference to its diameter, and is equal to acos(-1.0).

5.5.1 *e*

The number *e* is a mathematical constant that is the base of the natural logarithm. It's approximately equal to 2.71828182845904523536028747135266249775724709369995.

6. Statement

6.1 What is a statement

A statement is ended with semi-colon(';'). A statement is defined as a declaration and definition of variables or functions, an assignment, an execution of a function, or an input/ouput flow control.

6.2 Control flow

In CalCul ^{2}, we also provide if-else condition controls and while and for loops. They are all very easy to use.

If-Else:

```
if (bool_expr) {
   Statement 1;
}
else{
   Statement 2;
}
```

For If-Else Statement, if the boolean expression *bool_expr* holds, then the program will run *Statement 1*. Otherwise the program will run *Statement 2*.

loops:

While Statement:

```
while (bool_expr) {
   Statement;
}
```

The While Statement will keep running *Statement* until the boolean expression *bool_expr* doesn't hold.

For Statement:

```
for (val = a to/downto b) {
    Statement;
}
```

The For Statement will initially let *val* be *a* and keep running statement until *val* exceeds *b*. Everytime *val* will increase(or decrease if downto) by 1.

7. Declaration

7.1 Variable

A variable need not to be declared before assigning it a value, instead, a variable will be declare internally by the system when you assign a value to it. And the type of the variable is determined by its first definition.

For example, x=3, y=2.1,z=(1,2), x, y, z will be declared as type int, float, vector respectively. After first assignment, the type of a variable cannot be changed any more. That is to say, if x is assigned any value of type other than integer, an exception will be raised.

A variable name can be a sequence of letters, or a sequence of letters followed by a digit.

Example of valid variable names: x, a1, abc, abc2
Example of invalid variable names: _x, 1y, a2b

7.2 Function

A function declaration is a function name followed by a pair of parenthesis, inside which there should be the argument list. Function name can only be a letter or a sequence of letters. Note that a function has to be declared and defined at the same time as well, doing just either one is not allowed.

There are two way to declare and define a function:

1) A function declaration followed by '=' and expression, e.g. f(x)=x+1, f(x,y)=x*y
2) A function declaration followed by a pair of braces, inside which there should be statements, e.g. f(x1,x2){ statement1; statement2; }

8. System Functions

8.1 Math Functions

The math functions include *'sqrt','sin', 'cos', 'tan', 'asin', 'acos', 'atan', 'log'* and *'ln'*.
Double sqrt(Double): return the square root of the parameter.
Double sin(Double): return the sine value of the parameter. The unit of the parameter is radian.
Double cos(Double): return the cosine value of the parameter.
Double tan(Double): return the tangent value of the parameter.

Double asin(Double): return the inverse sine value of the parameter. The unit of the return value is radian.
Double acos(Double): return the inverse cosine value of the parameter.
Double atan(Double): return the inverse tangent value of the parameter.
Double log(Double): return the logarithm value of the parameter based on 10.
Double ln(Double): return the logarithm value of the parameter based on *e*.

9 Example

9.1 Define a variable and a function

```
## Define a variable
x = 3;

## Define a function f(x) and function variable x
f(x) = 3*x;
```

9.2 Evaluate a function

```
## Evaluate f(x) = 3*x + 2 such that x = 4, output: f(x)=14
f(x) = 3*x + 2;
:f(4);
```

9.3 Get Derivative of function

```
## Get derivative of function f(x)=3*x^2 , using " ' "
f(x) = 3*x^2;
:f'(x);          ## Output will be f'(x) = 6*x

## Get the derivative of function f(x1, x2)=2*x1+3*x2 with respect of x1
f(x1,x2) = 2*x1+3*x2;
:f'(x1);         ## Output will be f'(x1) = 2

## Get the gradient of function f(x1, x2)=2*x1+3*x2 when x1=2 and x2=3
f(x1,x2) = 2*x1+3*x2;
:f'(2,3);        ## Output will be f'(2,3) = [2,3];
```

9.4 Get Definite Integral of function

```
## Compute definite integral of f(x) within interval of 3 and 10
f(x) = 3*x^2  + x;
:f@x(3,10);          ## Output will be f@x(3,10)=1018.5
```

9.5 Recursion and Flow control

```
## using recursion and if-else control
f(x){
   if(x==0)
       f=0;
   else
       f=f(x-1)+x;
}
:f(3);          ## Output will be f(3)=6

## using loop control
f(x){
   f = 0;
   while(not x==0){
       f = f + x;
       x = x - 1;
    }
}
:f(3);          ## Output will be f(3)=6
:f'(3);         ## Error, the function f(x) is not a mathematical function
```