VC Language References Manual

Marissa Golden (mjg2238)
Carolyn Fine (crf2133)
Michelle Levine (mal2291)

Table of Contents

Introduction

We were inspired by codeacademy to create a programming language designed for teachers. Our goal is to make it easy for a teacher to program arithmetic tests with an auto-generated answer key. This language will be used to test students on the material they have already learned.

Lexical Conventions

Comments

There are two ways to comment your code in VC. The first way is single line comments which are started with `//` and are terminated with a new line. The second way to comment is with multi-line comments using `/* */`, where anything in between those characters is ignored by the scanner.

Reserved Keywords

- `MC`
- `eval`
- `fill_in`
- `TF`

Separators

In order to separate different statements a semicolon must be placed at the end of each statement. A colon is used after each question type in order to indicate that what follows is the expression.

Whitespace

We use the separators explained above to distinguish between statements. All whitespace (i.e. tabs and newlines) is ignored, but should be included for formatting as a good practice.

Types

VC supports the following data types:

MC = multiple choice

The MC data type is used to declare a mathematical expression that the compiler will then solve and autogenerate multiple choices, including the correct answer and 4 false answers. The student will then have to select the correct letter. One false answer will be generated by performing calculations from right to left, ignoring parentheses. One false answer will be generated by ignoring order of precedence going from left to right. One false answer will be generated by following most orders of precedence, but ignoring parentheses. One false answer may be an off by one error (or similar). The compiler will select the true answer and 3 out of the 4 false answers (at random) to display.

- Example: Teacher enters -> `MC: 2 * (3 + 7) + 12 / (2 + 2);`

  Compiler generates the following:

  a. 23 (correct answer)
  b. 26
  c. 14.5
  d. 21
  e. 24

  and selects ‘a’, ‘c’, ‘d’, and ‘e’ to display.

eval = evaluate the answer

The eval data type is used to declare a mathematical expression that the compiler will solve on the backend, but on the front end will leave a blank space for the user to input the correct answer.

- Example: Teacher enters -> `eval: 4 + 7 * 2;`

  Compiler generates -> 18
  Program displays -> 4 + 7 * 2 = ?

fill_in = fill in the blank

The fill_in data type is used to declare a mathematical expression that the compiler will evaluate and the program will replace one of the operands with ‘x’ for the display. The student will be prompted to solve for x.

- Example: Teacher enters -> `fill_in: 6 * 12 + 4 / 2;`

  Compiler generates -> 74
  Program displays -> 6 * 12 + x / 2 = 74
  x = _______

TF = true/false

The TF data type is used to declare a mathematical expression that the compiler will evaluate using the MC auto-generation process to either provide a True or False answer. The user will select a true/false checkbox.

- Example: Teacher enters -> `TF: 2 * (5 + 4) - 10 / (-2);`

Compiler generates -> -2 (generated ignoring order of precedence)
Program display -> 2 * (5 + 4) - 10 / (-2) = -2

- True
- False

Operators

Unary Operator

Symbol	Function
-	negative

Arithmetic Operators

Symbol	Function
(	open parenthesis
)	close parenthesis
^	exponent
*	multiplication
/	division
+	addition
-	subtraction

Note: In our language, there is no need for relational, equality or logical operators.

Conversions

By convention, all arithmetic expressions will be evaluated as floating points.

Expressions

```
expression → (expression)
           | expression ^ expression
           | expression * expression
           | expression / expression
           | expression + expression
           | expression - expression
           | - expression
```

An expression is formed by combining other expressions using arithmetic or unary operations. The above parser represents the only legal ways to make an expression.

Order of Operations

The precedence of evaluating expressions will follow the standard order of operations, e.g. PEMDAS.

Statements

An expression statement is formatted as follows:
`type: expression;`

Built-in Function

`page_break` - adds a new page to the display. Used to break up blocks of questions.

Example Code

```
//q1
MC: (30+2)/4-7+(6-4)*12;
//q2
TF: 3+4;
//q3
Fill_in: (9 * 4) + 2;
//q4
Eval: 12 - (3 + 5) * 2;
//q5
TF: 43 - 3 * 6;
```

Example Output

1. (30 + 2) / 4 - 7 + (6 - 4) * 12

a. 25
b. 36
c. -18.5
d. 30.5

2. 12 + 4 / 2 - (3 + 5) = 10

- True
- False

3. (9 * x) + 2 = 38
   x = ________

4. 12 - (3 + 5) * 2 = ?
   ____

5. 43 - 3 * 6 = 25

☐ True
☐ False