|  |
| --- |
|  |
| Simple Language Reference |
|  |
|  |
| **Andrew Laing** |
| **22/10/2017** |
|  |

Table of Contents

[The SIMPLE language. 2](#_Toc496470252)

[Compiling and running SIMPLE programs 2](#_Toc496470253)

[SIMPLE keyword reference 3](#_Toc496470254)

[REM 3](#_Toc496470255)

[END 3](#_Toc496470256)

[PRINT 3](#_Toc496470257)

[NEWLINE 3](#_Toc496470258)

[DIM 3](#_Toc496470259)

[LET 4](#_Toc496470260)

[INPUT 4](#_Toc496470261)

[GOTO 4](#_Toc496470262)

[GOSUB RETURN 4](#_Toc496470263)

[IF GOTO 4](#_Toc496470264)

[FOR NEXT STEP 5](#_Toc496470265)

[Examples of SIMPLE programs 5](#_Toc496470266)

[Source: inputTest.simple 5](#_Toc496470267)

[Source: arrayTest.simple 6](#_Toc496470268)

[Source: madlib.simple 6](#_Toc496470269)

[Source: stringComparisonTest.simple 7](#_Toc496470270)

[Source: stringComparisonTest1.simple 7](#_Toc496470271)

# The SIMPLE language.

The SIMPLE language is a simple yet powerful high-level language similar to an early version of the BASIC language. SIMPLE statements consist of a line number and a SIMPLE instruction. Line numbers must be in ascending order. SIMPLE allows the use of integers, integer arrays, character arrays, integer literals and string literals.

SIMPLE evaluates integer operations using the ***/***, ***\****, ***-***, ***+***, ***%*** operators, allows integer comparisons using ***<>***, ***<=***, ***>=***, ***==***, ***!=***, and string comparisons using ***==*** and ***!=.*** Operators in SIMPLE have the same precedence as in C.

When using variables remember that the SIMPLE compiler only recognises single alphabetic characters and character array variables are declared using the $ sign (e.g., ***a$***). The SIMPLE language is case insensitive so ***a*** is equivalent to ***A*** and this applies to the keywords, which will be described later.

SIMPLE has only one repetition structure, the ***for next*** loop. Structures like the C language’s ***while*** loop can be simulated using ***if/goto*** and ***goto*** statements.

# Compiling and running SIMPLE programs

SIMPLE language programs are compiled with the SimpleCompiler into files containing SML instructions which can then be run in the RunSML interpreter. To compile a SIMPLE language program simply move it to the directory containing the SimpleCompiler executable and run the following from the command line;

***SimpleCompiler -f filename.simple***

This will produce a file named ***out.sml*** which can then be run in the directory containing RunSML using the following command.

***RunSML -f out.sml***

# SIMPLE keyword reference

REM  
Used to include explanatory remarks in the source code of a program.  
Example usage:

*10 rem This is a comment  
20 rem Another comment*

END  
Terminates execution immediately.  
Example usage:

*10 print “A line of code”  
20 end*

PRINT  
Prints the value of the variable, integer literal, array element, character array, or string literal specified to the console. Note: variables can be printed separated by a comma, but strings and string literals can only be printed one at a time. The ***print*** statement does not add a newline character (use ***newline***)  
Example usage:

*10 print a, b, 20  
20 print a$  
30 print “Do you wish to continue (y/n) > ”*

NEWLINE  
Prints a newline to the console.  
Example usage:

*10 print “A string”  
20 newline*

DIM  
Declares and allocates storage space for one or more variables.  
Example usage:

*10 dim a(5) - declare an array to hold 5 integers  
20 dim a$(20) – declare a character array of 20 characters to hold a string*

LET  
Computes a value and assigns it to a new variable within the query.  
Example usage:

*20 let a = 20 \* 20  
30 let a$ = “A string”*

INPUT  
Inputs a value from the console and assigns it to a variable. Note: character arrays must be declared in advance using a ***dim*** statement.  
Example usage:

*10 dim a$(20)  
20 input a$  
30 input b*

GOTO  
Branches the flow of execution to a specified line number.  
Example usage:

*10 goto 60  
60 goto 20*

GOSUB RETURN  
Branches the flow of execution to a specified line number, then branches back to the line following that containing the ***gosub*** statement.  
Example usage:

*10 gosub 40  
20 print “carry on here”  
30 goto 70  
40 print “branched to here”  
50 return*

IF GOTO  
Conditionally branches the flow of execution to a specified line number, depending on the value of an expression.  
Example usage:

*10 if a == 10 goto 50  
20 if a == 20 goto 70*

FOR NEXT STEP  
Repeats a group of statements until an integer reaches a specified value, incrementing that value by step on each iteration. Note: the ***step*** keyword is optional, if omitted the step value will be one.  
Example usage:

*10 for I = 0 to 10 step 2  
20 print i  
30 next*

*10 for I = 0 to 5  
20 print i  
30 next*

Examples of SIMPLE programs   
  
Source: mathTest.simple

10 rem do some mathematics  
20 let a = 30 \* 9 / 5 + 32  
30 rem display the answer  
40 print "30 \* 9 / 5 + 32 = "  
50 print a  
60 newline  
70 end

## Source: inputTest.simple

5 print "Enter the value of x > "  
10 input x  
15 print "Enter the value of y > "  
20 input y  
30 let r = x \* y  
40 newline  
50 print "x \* y = "  
60 print r  
99 end

## Source: arrayTest.simple

10 dim a(3)  
20 let a(1) = 33  
30 let a(2) = 44  
40 let c = a(1) + a(2)  
45 print a(1)  
46 print " + "  
47 print a(2)  
48 print " = "  
50 print c  
55 newline  
60 end

## Source: madlib.simple

10 dim a$(20)  
20 dim b$(20)  
30 dim c$(20)  
40 print "Enter a male name > "  
50 input a$  
60 print "Enter an emotion > "  
70 input b$  
80 print "Enter an action > "  
90 input c$  
100 newline  
110 print a$  
120 print " was feeling "  
130 print b$  
140 print " so he decided to "  
150 print c$  
160 newline  
170 end

## Source: stringComparisonTest.simple

1 dim a$(20)  
2 dim b$(20)  
3 let a$ = "Frog"  
4 let b$ = "Frog"  
5 if a$ == b$ goto 9  
6 if a$ != b$ goto 11  
7 print "Comparison Failed"  
8 goto 12  
9 print "a$ == b$"  
10 goto 12  
11 print "a$ != b$"  
12 newline  
13 end

## Source: stringComparisonTest1.simple

10 dim a$(10)  
20 print "Enter a number > "  
30 input b  
35 let c = b  
40 let b = c \* c  
50 print "The square of "  
60 print c  
70 print " is "  
80 print b  
90 newline  
100 print "Do you want to enter another value? (y/n) > "  
110 input a$  
120 if a$ == "y" goto 20  
130 end