C Shell Script CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

Spring 2010

Revision Sheet

Date	Version	Revision Description	Author
05/12/10	1.0	Original Release	CSSE

1.0 CHECKLIST FOR SOURCE STATEMENT COUNTS

PHYSICAL AND LOGICAL SLOC COUNTING RULES

Measurement Unit	Order of Precedence	Physical SLOC	Logical SLOC	Comments
Executable lines	1	One per line	See table below	Defined in 2.8
Non-executable lines				
Declaration (Data) lines	2	One per line	See table below	Defined in 2.4
Compiler directives	3	One per line	See table below	Defined in 2.5
Comments				Defined in 2.7
On their own lines	4	Not included (NI)	NI	
Embedded	5	NI	NI	
Banners	6	NI	NI	
Empty comments	7	NI	NI	
Blank lines	8	NI	NI	Defined in 2.6

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*foreach*", "*while*" or "*if*" statement	1	Count once.	"*while*" is an independent statement.
R03	Statements ending by a semicolon or newline	2	Count once per statement, including empty statement.	

Table 2 Logical SLOC Counting Rules

2.0 DEFINITIONS

2.1 SLOC – Source Lines Of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

2.2 Physical SLOC – One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

2.3 Logical SLOC – Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

2.4 Data declaration line or data line – A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program.

2.5 Compiler directive – A statement that tells the compiler how to compile a program, but not what to compile. Bash shell script does not contain any compiler directives.

2.6 Blank line – A blank is a tab or space. What this actually means is - a blank is any chunk of white space between anything that is printable (a character or word). So a blank can be several spaces or tabs or a combination of multiples of the two.

2.7 Comment line – A comment is defined as a string of zero or more characters that follow a language-specific comment delimiter.

C shell script comment delimiters are "**#**". A whole comment line may span one line and does not contain any compilable source code. An embedded comment can co-exist with compilable source code on the same physical line. Banners and empty comments are treated as types of comments.

2.8 Executable line of code – A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, switch, case)
  - Iteration statements (foreach, while)
  - Empty statements (one or more ";")
  - Jump statements (goto, break, continue, exit)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements

  NOTE: See Section 3 of this document for examples of control statements.

- An executable line of code may not contain the following statements:
  - Whole line comments, including empty comments and banners
  - Blank lines

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

EXECUTABLE LINES				
SELECTION STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

ESS1	if, else if, else and nested if statements	if (<*boolean expression*>) then
		  <*statements*>
		endif
		
		if (<*boolean expression*>) then
		  <*statements*>
		else
		  <*statements*>
		endif
		
		if (<*boolean expression*>) then
		  <*statements*>
		else if (<*boolean expression*>)
		  <*statements*>
		else
		  <*statements*>
		endif
		
		if (<*boolean expression*>) <*statement*>
		
		NOTE: complexity is not considered, i.e. multiple "&&" or "||" as part of the expression.
			if ($x != 0) then
			  echo "non-zero"
			endif
			
			if ($x > 0) then
			  echo "positive"
			else
			  echo "negative"
			endif
			
			if ($x == 0) then
			  echo "zero"
			else if ($x > 0) then
			  echo "positive"
			else
			  echo "negative"
			endif
			
			if ($x != 0) echo "non-zero"
				1
				1
				0
				
				1
				1
				0
				1
				0
				
				1
				1
				1
				1
				0
				1
				0
				
				2

ESS2	switch and nested switch statements	switch (<*string*>)
		  case <*string 1*>:
		  <*statements*>
		  breaksw
		  case <*string 2*>:
		  <*statements*>
		  breaksw
		  case <*string 3*>:
		  <*statements*>
		  break
		  default:
		  <*statements*>
		endsw
			switch ($str)
			  case "1":
			  echo "one"
			  breaksw
			  case "2":
			  echo "two"
			  breaksw
			  case "3":
			  echo "three"
			  breaksw
			  default:
			  echo "invalid case"
			endsw
				1
				0
				1
				1
				0
				1
				1
				0
				1
				1
				0
				1
				0

ITERATIONS STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EIS1	foreach	foreach <*name*> (<*wordlist*>) <*statements*> end	foreach i ($d) echo $i end	1 1 0

EIS2	while	while <*expression*>
		  <*statements*>
		end
			while ($j <= 10)
			  echo '2 **' $j = $i
			  @ i *= 2
			  @ j++
			end
				1
				1
				1
				1
				0

JUMP STATEMENTS (ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EJS1	goto, label	goto *label* . . label:	loop1: $x++ if ($x < $y) goto loop1	0 1 2
EJS2	break	break	if ($i > 10) break	2
EJS3	exit function	exit *return_code*	if ($x < 0) exit 1	2

EJS4	continue	continue	while ($i < 10)
			  $i++
			  if ($i == 5) then
			  continue
			  else
			  $j++
			  endif
			end
				1
				1
				1
				1
				0
				1
				0
				0

EXPRESSION STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EES1	function call	<*function_name*> (<*parameters*>)	read_file (name)	1
EES2	assignment statement	$<name> = <value>	$a = “value”	1
EES3	empty statement (is counted as it is considered to be a placeholder for something to call attention)	one or more “;” in succession	;	1 each