Fortran CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

March, 2010

Revision Sheet

Date	Version	Revision Description	Author
12/03/07	1.0	Original Release	CSSE
03/23/10	1.1	UCC Update	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 2.9
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	
Empty comments	6	NI	NI	
Blank lines	7	NI	NI	Defined in 2.7

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*do*-x", "while-do" combination, "*if*", "elseif" statement	1	Count once per structure	
R02	Compiler directive	2	Count once per directive	
R03	data declaration and data assignment	3	Count once per declaration/assignment	
R04	Jump statement	4	Count once per keyword	
R05	Function/Subroutine call	5	Count once per call	
R06	Semicolon in statement	6	Count once per semicolon	
R07	Multiple statements in a line	7	Count one per executable 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.

It includes job control language (compiler directive), format statements, and data declarations (data lines). Thus, a line containing two or more source statements count as one physical SLOC; a five line data declaration counts as five physical SLOCs.

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. In the case of FORTRAN, there is no semicolon terminating every single executable line.

The number of logical SLOC within a source file is defined to be the sum of the number of logical SLOCs classified as compiler directives, data lines, or executable lines. It excludes comments (whole or embedded) and blank lines. Thus, a line containing two or more source statements count as multiple logical SLOCs. A single logical statement that extends over five physical lines counts as one logical SLOC. Specifically, the logical SLOC found within a file containing software written in the FORTRAN 77 programming language is equivalent to the number of physical lines less the number of physical continuation lines. The logical SLOC found within a file containing software written in the FORTRAN 90 programming language may be computed by
(1) counting the number of separator semicolons, i.e., non-terminal semicolons used to separate multiple logical statements on the same physical line,
(2) adding the number of physical lines,
(3) subtracting the number of physical continuation lines, then
(4) subtracting the number of physical lines that only contain the following keywords USE, CYCLE, CASE, CONTAINS, PUBLIC, PRIVATE, ELSE END INTERFACE, END TYPE, END CASE, END IF, END DO, END SELECT, END WHERE, END SUBROUTINE, END FUNCTION, END PROGRAM, and END MODULE.
The logical SLOC definition was selected due to
(1) compatibility with parametric software cost modeling tools, and
(2) ability to support software metrics collection.

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.

The following table lists Fortran keywords that denote data declaration lines:

Data Types	FORTRAN 77	FORTRAN 90
SUBROUTINE	X	X
FUNCTION	X	X
VIRTUAL	X	
BLOCK DATA	X	

DOUBLE COMPLEX	X	
COMMON	X	X
NAMELIST	X	X
IMPLICIT	X	X
INTEGER	X	X
REAL	X	X
COMPLEX	X	X
CHARACTER	X	X
DATA	X	X
PARAMETER	X	X
DIMENSION	X	X
DOUBLE PRECISION	X	X
EQUIVALENCE	X	X
RECORD	X	
STRUCTURE	X	
BYTE	X	
LOGICAL	X	X
EXTERNAL	X	X
INTRINSIC	X	X
UNION	X	
MAP	X	
VOLATILE	X	
SAVE	X	X
PROGRAM		X
MODULE		X
CONTAINS		X
ASSIGN		X
USE		X
TYPE	X	X
INTERFACE		X
ALLOCATE		X
REALLOCATE		X
DEALLOCATE		X
NULLIFY		X
OPTIONAL		X
PUBLIC		X
PRIVATE		X
POINTER		X
TARGET		X

Table 3 Data Type Keywords

NOTE: See Section 3 of this document for examples of data declaration lines.

Execute Keywords	**FORTRAN 77**	**FORTRAN 90**
GOTO	X	X
IF	X	X
ELSE WHERE		X
ELSE IF		X
ELSE		X
DO	X	X
CYCLE		X
SELECT		X
CASE		X
WHERE		X
CALL	X	X
FORMAT	X	X
READ	X	X
PRINT	X	X

Table 4 Executable Statements Keywords

NOTE: See Section 3 of this document for examples of executable lines.

2.5 Compiler directive - A statement that tells the compiler how to compile a program, but not what to compile.

A list of common Fortran directives is presented in the table below:

Compiler Directive Keywords	FORTRAN 77	FORTRAN 90
OPTIONS	X	X
INCLUDE	X	X
DICTIONARY	X	X

Table 5 Compiler Directive Keywords

NOTE: See Section 3 of this document for examples of compile directive lines.

2.6 Blank line – A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

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

There are several ways to indicate comments in Fortran. The table below illustrates the different methods.

COMMENTS	
Rule	Example
A line that begins with the letter "c" or “C” or an asterisk “*” in the first column of the line	C This is a comment c This is a comment * This is a comment
All characters following an exclamation mark “**!**”, except in a character string, are comments.	Year = Year + 1 ! This is a comment
Blank lines are not counted	

Table 6 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)
  - Iteration statements (do-enddo)
  - Jump statements (return, goto, break, continue, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)

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

- An executable line of code may not contain the following statements:
  - Compiler directives
  - Data declaration (data) lines
  - Whole line comments, including empty comments
  - 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 (*logical expression*) *executable statement*
		
		or
		
		if (*logical expression*) then
		  statements
		else
		  statements
		endif
		
		or
		
		if (*logical expression*) then
		  statements
		elseif (*logical expression*) then
		  statements
		  :
		  :
		  :
		  :
		else
		  statements
		endif
			if (x .lt. 0) x = -x
			
			or
			
			if (x .ge. y) then
			  write(*,*) 'x'
			else
			  write(*,*) 'x'
			endif
			
			or
			
			if (x .gt. 0) then
			  if (x .ge. y) then
			  write(*,*) 'x'
			  else
			  write(*,*) 'x'
			  endif
			elseif (x .lt. 0) then
			  write(*,*) 'x is neg'
			else
			  write(*,*) 'x is zero'
			endif
				2
				
				1
				1
				0
				1
				0
				
				1
				1
				1
				0
				1
				0
				1
				1
				0
				1
				0

ESS2	select and nested select statements	Datatype :: selector
		
		select case (selector)
		  case (label-list-1)
		  statements-1
		  case (label-list-2)
		  statements-2
		  :
		  :
		  case (label-list-n)
		  statements-n
		  case default
		  statements-default
		end select
			integer :: Class
			
			select case (Class)
			  case (1)
			  write(*,*) 'Freshman'
			  case (2)
			  write(*,*) 'Sophomore'
			  case (3)
			  write(*,*) 'Junior'
			  case (4)
			  write(*,*) 'Senior'
			  case default
			  write(*,*) "no class"
			end select
			
			write(*,*) 'Done'
				1
				
				1
				0
				1
				0
				1
				0
				1
				0
				1
				0
				
				1

ITERATIONS STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

EIS1	do-enddo	do *label var = expr1, expr2, expr3*
		  statements
		label continue
		
		or
		
		do *var = expr1, expr2, expr3*
		  statements
		enddo
		
		or
		
		do
		  statements
		  if *(logical expr)* exit
		  statements
		enddo
		
		where *expr1* specifies the initial value of *var*, *expr2* is the terminating bound, and *expr3* is the increment (step).
			do 20 i = 10, 1, -2
			  write(*,*) 'i =', i
			20 continue
			
			or
			
			do i = 10, 1, -2
			  write(*,*) 'i =', i
			enddo
			
			or
			
			i = 10
			do
			  write(*,*) 'i =', i
			  if (i < 1) exit
			  i = i - 2
			enddo
				1
				1
				1
				
				1
				1
				0
				
				1
				1
				1
				2
				1
				0

EIS2	do-while	do *label* while *(logical-expr)*
		  statements
		label continue
		
		or
		
		do while *(logical expr)*
		  statements
		enddo
			do 20 while (i == 10)
			  write(*,*) 'i =', i
			20 continue
			
			or
			
			do while (i == 10)
			  write(*,*) 'i =', i
			enddo
				1
				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	*label* if *(logical expr)* then
		  statements
		  :
		  goto *label*
		endif
			10 if (n .le. 100) then
			  n = 2 * n
			  write (*,*) n
			  goto 10
			endif
				1
				1
				1
				1
				0

EJS2	cycle	cycle *label*
		
		Or
		
		cycle
			outer: do i = 1, n
			middle: do j = 1, m
			inner: do k = 1, l
			  :
			  :
			  :
			if (a(i, j, k) < 0) exit outer
				1
				1
				1
				
				2

EXPRESSION STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EES1	function call or procedure call	<function_name> (<parameters>) call <subroutine_name> (<parameters>)	cos(4) call avg(a, b, c)	1 1
EES2	assignment statement	<name> = <value> <name> = <value>; <name> = <value>; …	a = 174.5 a = 2; b = 7; c = 3	1 3
EES3	empty statement	one or more ";" in succession	a = 2;	1 per each

DECLARATION (DATA) LINES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

DDL1	function declaration	*type* function *name (list-of-variables)*
		  declarations
		  statements
		  :
		  :
		  return
		end
			function fact(n)
			  fact = 1
			  do 10 j = 2, n
			  fact = fact * j
			  10 continue
			  return
			end
				1
				1
				1
				1
				1
				1
				0

	subroutine declaration	subroutine *name (list-of-arguments)*
		  declarations
		  statements
		  return
		end
			subroutine iswap(a, b)
			  integer a, b
			  integer tmp
			  tmp = a
			  a = b
			  b = tmp
			  return
			end
				1
				1
				1
				1
				1
				1
				1
				0

	variable declaration	<type> <name>	real a	1
	type declaration	type ( )	type (person) chairman	1

COMPILER DIRECTIVES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
CDL1	directive types	options *options*	options /CHECK=ALL/F77	1
		include *character-literal-constant*	include 'my_common_blocks'	1
		dictionary *character-literal-constant*	dictionary 'salary'	1