Fortran Code Count™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
2/19/2016	1.0	Original Release	Matthew Swartz
3/9/2016	1.1	Added Complexity Rules and Appendix for differences between UCC and UCC-G	Madhvi Kansagara
7/8/2016	1.2	Updated complexity table	Matthew Swartz
Date	Version No.	Click here to Description.	Name

Table of Contents

1. Definitions

1.1. **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.

1.2. **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.

1.3. **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, and 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 free format Fortran programming language (.f90, .F90, .f95, .F95, .f03, .F03, .hpf) is equivalent to the number of physical lines less the number of physical continuation lines.

1.4. **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 the Fortran keywords that denote data declaration lines:

double precision	equivalence	select type	deallocate	reallocate
subroutine	associate	character	dimension	interface
intrinsic	parameter	recursive	allocate	contains
external	function	implicit	namelist	optional
complex	generic	integer	logical	nullify
program	assign	common	import	module
final	data	enum	real	save
type	use			

Table 1 Fortran Data Keywords

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile.

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

!$pragma sparc	!$pragma sun	#dictionary	#options
#include	!$pragma	#ifndef	#define
#ifdef	#undef	#endif	!dir$&
!$par&	!mic$&	#else	#elif
!dir$	!$omp	!$par	!mic$
#if			

Table 2 Fortran Compiler Directive

1.6. **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).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter. All characters following an exclamation mark "!", except in a character string, are comments.

1.8. **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.)
- Block statements

An executable line of code may not contain the following statements:

- Compiler directives
- Data declaration (data) lines
- Whole line comments, including empty comments and banners
- Blank lines

2. Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES			
MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable lines	1	One per line	Defined in 1.8
Non-executable lines			
Declaration (Data) lines	2	One per line	Defined in 1.4
Compiler directives	3	One per line	Defined in 1.5
Comments			Defined in 1.7
On their own lines	4	Not included (NI)	
Embedded	5	NI	
Banners	6	NI	
Empty comments	7	NI	
Blank lines	8	NI	Defined in 1.6

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	

3. Examples

EXECUTABLE LINES

SELECTION Statement

ESS1 - if, else if, else and nested if statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

Datatype :: selector
 	
select case (selector)
  case (label-list-1)
  statements-1
  case (label-list-2)
  statements-2
	integer :: Class
	
	select case (Class)
	  case (1)
	  write(*,*) 'Freshman'
	  case (2)
	  write(*,*) 'Sophomore'
		1
		
		1
		0
		1
		0
		1

: : case (label-list-n) *statements-n* case default *statements-default* end select	case (3) write(*,*) 'Junior' case (4) write(*,*) 'Senior' case default write(*,*) "no class" end select	0 1 0 1 0 1 0

ITERATION Statement

EIS1 – do-enddo

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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 Statement

EJS1 – goto label

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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
	if (j == 5) cycle middle
	if (i == 5) cycle
	  :
	  :
	  :
	enddo inner
	enddo middle
	enddo outer
		1
		1
		1
		
		2
		2
		2
		
		0
		0
		0

EJS3 - Exit		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
exit *label* *or* exit	exit 10 or exit	1 1
EJS4 – continue		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

label continue
  statements
if *(logical expr)* goto *label*
	20 continue
	  k = 3
	if *(k==3)* goto *20*
		1
		1
		2

EXPRESSION Statement		
EES1 - function call or procedure call		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<function_name> (<parameters>) call <subroutine_name> (<parameters>)	cos(4) call avg(a, b, c)	1 1
EES2 - assignment statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<name> = <value> <name> = <value>; <name> = <value>; …	a = 174.5 a = 2; b = 7; c = 3	1 3
EES3 – empty statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more “;” in succession	a = 2;	1 per each

DECLARATION OR DATA LINES

DDL1 - function declaration subroutine declaration variable declaration type declaration

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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 *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

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

COMPILER DIRECTIVES

CDL1 – directive type

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
package <package_name>; import <package_name>;	package test; import java.io*;	1 1

4. Complexity

Complexity measures the occurrences of different keywords in code baseline. Below table identifies the categories and their respective keywords that are counted as part of the complexity metrics.

Table 3 - Complexity Keywords List

Math Functions	Trig	Log	Calculations	Conditionals	Logic	Pre-processor	Assignment	Pointer
dot_product	acosh	log10	**	select case	.false.	!$pragma sparc	=	=>
ceiling	asinh	log	+	select type	.neqv.	!$pragma sun		
matmul	atan2		-	else if	.true.	#dictionary		
modulo	atanh		/	elseif	.and.	#options		
floor	acos		*	forall	.not.	#include		
conjg	asin			else	.eqv.	!$pragma		
dprod	atan			do	.eq.	#ifndef		
sign	cosh			if	.ne.	#define		
sqrt	sinh				.lt.	#ifdef		
mod	tanh				.gt.	#undef		
abs	cos				.le.	#endif		
dim	sin				.ge.	!dir$&		
exp	tan				.or.	!$par&		
max					==	!mic$&		
min					!=	#else		
					<=	#elif		
					>=	!dir$		
					&&	!$omp		
					||	!$par		
					/=	!mic$		
					<	#if		
					>			