Ada CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

July, 2007

Revision Sheet

Date	Version	Revision Description	Author
07/25/07	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	
Blank lines	6	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	Statements ending by a semicolon	1	Count once per statement, including empty statement.	Semicolons as part of parameter list in function, procedure, or task entry definition is not counted.

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#, Ada) 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.

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

`program`	`body`	`subtype`	`renames`
`function`	`private`	`array`	`limited`
`package`	`separate`	`record`	`use`
`task`	`constant`	`access`	`with`
`generic`	`type`	`declare`	`new`

Table 3 Data Declaration Types

NOTE: See Section 3 of this document for examples of data declaration 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 Ada directives is presented in the table below:

`pragma`	`interface`	`pack`	`storage_unit`
`controlled`	`list`	`page`	`suppress`
`elaborate`	`memory_size`	`priority`	`system_name`
`inline`	`optimize`	`shared`	

Table 4 Compiler Directives

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.

Ada comment delimiter is "`--`". A whole comment line may span one or more lines and does not contain any compilable source code. An embedded comment can co-exist with compilable source code on the same physical line.

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, case)
  - Iteration statements (loop)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, exit)
  - Expression statements (function calls, procedure 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:
  - Compiler directives
  - Data declaration (data) lines
  - Whole line 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-elsif-else statements	if <boolean expression> then
		   <statements>
		end if;
		
		if <boolean expression> then
		   <statement>
		else
		   <statement>
		end if;
		
		if <boolean expression> then
		   <statements>
		elsif <boolean expression> then
		   <statements>
		…
		else
		   <statements>
		end if;
		
		NOTE: complexity is not considered, i.e. multiple “and” or “or” as part of the expression.
			if x /= 0 then
			   Put_Line (“non-zero”);
			end if;
			
			if x > 0 THEN
			   Put_Line (“positive”);
			else
			   Put_Line (“negative”);
			end if;
			
			if x = 0 then
			   Put_Line (“zero”);
			elsif x > 0 then
			   Put_Line (“positive”);
			else
			   Put_Line (“negative”);
			end if;
			
			if x /= 0 and x > 0 then
			   Put (x);
			end if;
				0
				1
				1
				
				0
				1
				0
				1
				1
				
				0
				1
				0
				1
				0
				1
				1
				
				0
				1
				1

ESS2	case statements	case <expression> is
		   when <choice1> =>
		      <statements>
		   when <choice2> =>
		      <statements>
		   …
		   when <choiceN> =>
		      <statements>
		   when others =>
		      <statements>
		end case;
			case number is
			   when 1 | 11 =>
			      foo1();
			   when 2 =>
			      foo2();
			   when 3: =>
			      foo3();
			   when others =>
			      Put_Line (“invalid”);
			end case;
				0
				0
				1
				0
				1
				0
				1
				0
				1
				1

ESS3	exception statements	exception
		   when <exception_choice1> =>
		      <statements>
		   when <exception_choice2> =>
		      <statements>
		   …
		   when others =>
		      <statements>
		end;
			exception
			   when Constraint_Error =>
			      Put_Line (“range error”);
			   when Storage_Error =>
			      Put_Line (“out of RAM”);
			   when others =>
			      Put_Line (“other error”);
			      raise; -- raise exception
			end;
				0
				0
				1
				0
				1
				0
				1
				1
				1

ITERATIONS STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

EIS1	simple loop	loop
		    <statements>
		end loop;
			loop
			    null;
			end loop;
				0
				1
				1

EIS2	while-loop	while <boolean expression> loop
		    <statements>
		end loop;
			while i < 10 loop
			    Put (i);
			    i := i + 1;
			end loop;
				0
				1
				1
				1

EIS3	for-loop	for <loop counter> in <range> loop
		    <statements>
		end loop;
			for i in 1 .. 5 loop
			    Put (i);
			end loop;
				0
				1
				1

JUMP STATEMENTS
(ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

EJS1	return	return <expression>;	if i = 0 then
			    return null;
			end if;
				0
				1
				1

EJS2	goto, label	goto *label*;
		…
		<<*label*>>
			<<loop1>>
			    x := x + 1;
			    if (x < y) then
			        goto loop1;
			    end if;
				0
				1
				0
				1
				1

EJS3	exit	exit;
		
		exit when <boolean expression> ;
			loop
			    if x < 0 then
			        exit;
			    end if;
			end loop;
			
			loop
			    exit when x < 0;
			end if;
				0
				0
				1
				1
				1
				
				0
				1
				1

EXPRESSION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EES1	function and procedure call	<func_name> [(<params>)];	Put_Line (name);	1

EES2	assignment statement	<name> := <value>;	x := y;
			a := 1; b := 2; c := 3;
				1
				3

EES3	empty statement (is counted as it is considered to be a placeholder for something to	one or more “;” in succession	;	1 per each

BLOCK STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EBS1	simple block	-- start of block	-- start of block	0
	(related	begin	begin	0
	statements	<statements>	Put _Line (“Hello”);	1
	treated as a	end;	end;	1
	unit)	-- end of block	-- end of block	0
	procedure	procedure <proc_name> [(<params>)] is	procedure foo (i : in	0
	definition	<declarations>	Integer) is	0
		begin	begin	0
		<statements>	Put (i);	1
		end [<proc_name>];	end foo;	1
	function	function <func_name> [(<params>)]	function sum (a, b : in	0
	definition	return <ret_type> is	Float) return Float is	0
		<declarations>	begin	0
		begin	return a + b;	1
		<statements>	end sum;	1
		end [<func_name>];		
	task definition	task body <task_name> is	task body activity is	0
		<declarations>	begin	0
		begin	loop	0
		<statements>	exit;	1
		end [<task_name>];	end loop;	1
			end;	1
	package	package body <pkg_name> is	package body foo_pkg is	0
	definition	<declarations>	begin	0
		begin	procedure foo_proc is	0
		<statements>	begin	0
		end [<pkg_name>];	Put_Line(“Foo Pkg”);	1
			end foo_proc;	1
			end;	1

DECLARATION (DATA) LINES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
DDL1	procedure	procedure <proc_name> [(<params>)];	procedure foo (p : in	0
	specification		Integer);	1
	function	function <func_name> [(<params>)]	function foo return Integer;	1
	specification	return <ret_type>;		
	task	task <task_name>;	task action;	1
	specification			
	package	package <pkg_name> is	package foo is	0
	specification	<declarations>	procedure foo1 (x : Float);	1
		end [<pkg_name>];	function foo2 (x : Integer;	0
			y : Float)	0
			return Float;	1
			end area;	1
	enumeration	type <name> is (<enumeration_list>);	type answer is (‘y’, ‘n’);	1
	type definition			
	subtype	subtype <type_name> is <type>	subtype digits is Integer	0
	definition	range <discrete_range>;	range 0 .. 9;	1
	record	type <name> is	type position is	0
	definition	record	record	0
		<record structure>	x : Integer;	1
		end record;	y : Integer;	1
			end record;	1
	variable	declare	declare	0
	declaration	<name> : <type>;	amount, price : Float;	1
			index : Integer;	1
	task entry	entry <entry_name> [(<params>)];	entry foo;	1

COMPILER DIRECTIVES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
CDL1	directive types	pragma <name> [(<params>)];	pragma Export (C, foo, “foo”);	1