Ada CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

July , 2007

Revision Sheet

Date	Version	Revision Description	Author
7/25/2007	1.0	Original Release	CSSE
1/2/2013	1.1	Updated document template	CSSE

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#, Ada) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

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 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 1 Data Declaration Types

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 Ada keywords that denote compiler directive lines:

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

Table 2 Compiler Directives

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.

Ada comment delimiters are "--". A whole comment line may span one line and does not contain any compliable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of 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, case)
  - Iteration statements (loop)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, exit)
  - 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	
Embedded	5	Not Included	
Blank lines	6	Not Included	Defined in 1.6

LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	Statements ending with 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

3. Examples

EXECUTABLE LINES

SELECTION Statements

ESS1 – if-elsif-else statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if <boolean expression> then
  <statements>
end if;
	if x /= 0 then
	Put_Line (“non-zero”);
	end if;
		0
		1
		1

if <boolean expression> then
  <statement>
else
  <statement>
end if;
	if x > 0 THEN
	Put_Line (“positive”);
	else
	Put_Line (“negative”);
	end if;
		0
		1
		0
		1
		1

if <boolean expression> then
  <statements>
elsif <boolean expression> then
  <statements>
…
else
  <statements>
end if;
	if x = 0 then
	Put_Line (“zero”);
	elsif x > 0 then
	Put_Line (“positive”);
	else
	Put_Line (“negative”);
	end if;
		0
		1
		0
		1
		0
		1
		1

NOTE: complexity is not considered, i.e. multiple “and” or “or” as part of the expression.	if x /= 0 and x > 0 then
	Put (x);
	end if;
		0
		1
		1

ESS2 – case statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
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

ITERATION Statements

EIS1 – Simple Loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
loop <statements> end loop;	loop null; end loop;	0 1 1

EIS2 – While Loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
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

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
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)

EJS1 - return

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

EJS2 – goto, label

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

EJS3 - exit

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

exit;	loop
	if x < 0 then
	exit;
	end if;
	end loop;
		0
		0
		1
		1
		1

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

EXPRESSION Statements

EES1 – function and procedure call

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<func_name> [(<params>)];	Put_Line (name);	1

EES2 – assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<name> := <value>;	x := y; a := 1; b := 2; c := 3;	1 3

EES3 – empty statement (is counted and considered to be a placeholder for something to call attention)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more ";" in succession	;	1 per each

BLOCK Statements

EBS1 – simple block (related statements treated as a unit)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

-- start of block
begin
<statements>
end;
-- end of block
	-- start of block
	begin
	Put _Line (“Hello”);
	end;
	-- end of block
		0
		0
		1
		1
		0

EBS1 – procedure definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
procedure <proc_name> [(<params>)] is <declarations> begin <statements> end [<proc_name>];	procedure foo (i : in Integer) is begin Put (i); end foo;	0 0 0 1 1

EBS1 – function definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
function <func_name> [(<params>)] return <ret_type> is <declarations> begin <statements> end [<func_name>];	function sum (a, b : in Float) return Float is begin return a + b; end sum;	0 0 0 1 1

EBS1 – task definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

task body <task_name> is
<declarations>
begin
<statements>
end [<task_name>];
	task body activity is
	begin
	loop
	exit;
	end loop;
	end;
		0
		0
		0
		1
		1
		1

EBS1 – package definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

package body <pkg_name> is
<declarations>
begin
<statements>
end [<pkg_name>];
	package body foo_pkg is
	begin
	procedure foo_proc is
	begin
	Put_Line("Foo Pkg");
	end foo_proc;
	end;
		0
		0
		0
		0
		1
		1
		1

DECLARATION OR DATA LINES

DDL1 – procedure specification

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
procedure <proc_name> [(<params>)];	procedure foo (p : in Integer);	0 1

DDL1 – function specification

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
function <func_name> [(<params>)] return <ret_type>;	function foo return Integer;	1

DDL1 – task specification

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
task <task_name>;	task action;	1

DDL1 – package specification

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
package <pkg_name> is <declarations> end [<pkg_name>];	package foo is procedure foo1 (x : Float); function foo2 (x : Integer; y : Float) return Float; end area;	0 1 0 0 1 1

DDL1 – enumeration type definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
type <name> is (<enumeration_list>);	type answer is ('y', 'n');	1

DDL1 – subtype definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
subtype <type_name> is <type> range <discrete_range>;	subtype digits is Integer range 0 .. 9;	0 1

DDL1 – record definition

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
type <name> is record <record structure> end record;	type position is record x : Integer; y : Integer; end record;	0 0 1 1 1

DDL1 – variable declaration

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
declare <name> : <type>;	declare amount, price : Float; index : Integer;	0 1 1

DDL1 – task entry

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
entry <entry_name> [(<params>)];	entry foo;	1

COMPILER DIRECTIVES

CDL1 – directive types

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
pragma <name> [(<params>)];	pragma Export (C, foo, "foo");	1

4. Notes on Special Character Processing

1) Quotes: Quotes are of three types
Start of Quotes: "\""
End of Quotes: "\""
Escape Rear Quotes: '\"'

2) Four types of file extensions are recognized for Ada: .ada, .a, .adb, .ads