C/C++ CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

June, 2007

Revision Sheet

Date	Version	Revision Description	Author
6/22/07	1.0	Original Release	CSSE
10/16/07	1.1	Updated the example of *switch* statement	Vu Nguyen

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 2.8
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 2.7

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*for*", "*while*" or "*if*" statement	1	Count once.	"*while*" is an independent statement.
R02	*do* {…} *while* (…); statement	2	Count once.	Braces *{…}* and semicolon *;* used with this statement are not counted.
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement.	Semicolons within "*for*" statement are not counted. Semicolons used with R01 and R02 are not counted.
R04	Block delimiters, braces {…}	4	Count once per pair of braces *{..}*, except where a closing brace is followed by a semicolon, i.e. *};* or an opening brace comes after a keyword "*else*".	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by *{…}*.
R05	Compiler directive	5	Count once per directive.	

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.

The following table lists C/C++ keywords that denote data declaration lines:

Simple Data Types	Compound and User Defined Data Types	Access Specifiers	Type Qualifiers
`bool`	`class`	`private`	`const`
`char`	`struct`	`protected`	`volatile`
`double`	`union`	`public`	
`float`	`enum`	`friend`	
`int`	`typedef`	**Storage Class Specifiers**	**Miscellaneous**
`long`	`template`	`auto`	`asm`
`short`	`typename`	`extern`	`explicit`
`signed`		`mutable`	`inline`
`unsigned`		`register`	`namespace`
`void`		`static`	`using`
`wchar_t`			`operator`
			`virtual`

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 C/C++ directives is presented in the table below:

`#define`	`#ifndef`	`#include`	`#dictionary`
`#undef`	`#else`	`#line`	`#module`
`#if`	`#elif`	`#pragma`	`#import`
`#ifdef`	`#endif`	`#error`	`#using`

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.

C/C++ comment delimiters are "//" and "/*". 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. 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, *?* operator, switch)
  - Iteration statements (for, while, do-while)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, break, continue, exit function)
  - 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:
  - Compiler directives
  - Data declaration (data) lines
  - 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	if (<boolean expression>)	if (x != 0)	1
	and	<statements>;	printf (“non-zero”);	1
	nested if			
	statements	if (<boolean expression>) <statement>;	if (x > 0) printf (“positive”);	2
		else <statement>;	else printf (“negative”);	1
		if (<boolean expression>)		
		<statements>;	if (x == 0)	1
		else if (<boolean expression>)	printf (“zero”);	1
		<statements>;.	else if (x > 0)	1
		.	printf (“positive”);	1
		.	else	0
		else <statements>;	printf (“negative”);	1
			if ((x != 0) && (x > 0))	1
			printf (“%d”, x);	1
		if (<boolean expression>)		
		{	if (x != 0)	1
		<statements>;	{	0
		}	printf (“non-zero”);	1
		else	}	0
		{	else	0
		<statements>;	{	0
		}	printf (“zero”);	1
			}	0
		NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.		
ESS2	? operator	Exp1?Exp2:Exp3	x > 0 ? printf (“+”) : printf (“-”);	1
ESS3	switch and	switch (<expression>)	switch (number)	1
	nested switch	{	{	0
	statements	case <constant 1> :	case 1:	0
		<statements>;	case 11:	0
		break;	foo1();	1
		case <constant 2> :	break;	1
		<statements>;	case 2:	0
		break;	foo2();	1
		case <constant 3> :	break;	1
		<statements>;	case 3:	0
		break;	foo3();	1
		default	break;	1
		<statements>;	default	0
		}	printf (“invalid case”);	1
			}	0
ESS4	try-catch	try	try	1
		{	{	0
		// code that could throw	cout << "Calling func \n";	1

		// an exception	MyFunc();	1
		}	}	0
		catch (exception-declaration)	catch (IOException e)	1
		{	{	0
		// code that executes when	cout << “Error: “ << e;	1
		// exception-declaration is thrown	}	0
		// in the try block		
		}		

ITERATIONS STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
EIS1	for	for (initialization; condition; increment) *statement*;	for (i = 0; i < 10; i++)	1
			printf (“%d”, i);	1
			for (i = 0; i < 10; i++)	1
			{	0
			printf (“%d”, i);	1
			}	0
		NOTE: “for” statement counts as one, no matter how many optional expressions it contains, i.e. for (i = 0, j = 0; i < 5, j < 10; i++, ,j++)		
EIS2	empty statements (could be used for time delays)	for (i = 0; i < SOME_VALUE; i++) ;	for (i = 0; i < 10; i++) ;	2
EIS3	while	while (<boolean expression>) <statement>;	while (i < 10)	1
			{	0
			printf (“%d”, i);	1
			i++;	1
			}	0
EIS4	do-while	do	do	0
		{	{	0
		<statements>;	ch = getchar();	1
		} while (<boolean expression>);	} while (ch != ‘\n’);	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) return;	2
EJS2	goto, label	goto *label*;	loop1:	0
		.	x++;	1
		.	if (x < y) goto loop1;	2
		label:		
EJS3	break	break;	if (i > 10) break;	2
EJS4	exit function	void exit (int return_code);	if (x < 0) exit (1);	2
EJS5	continue	continue;	while (!done)	1
			{	0
			ch = getchar();	1
			if (char == ‘\n’)	1
			{	0
			done = true;	1

DECLARATION (DATA) LINES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
DDL1	function prototype,	<type> <name> (< parameter_list>);	void foo (int param);	1
	variable declaration,	<type> <name>;	double amount, price;	1
			int index;	1
	struct declaration	struct <name>	struct S	0
		{	{	0
		<type> <name>;	int x;	1
		<type> <name>;	int y;	1
		}	};	1
		struct	struct	0
		{	{	0
		<type> <name>;	int x;	1
		<type> <name>;	int y;	1
		} <name>;	} S;	2
	typedef	typedef <type> <name>;	typedef int MY_INT;	1
		typedef struct <name>	typedef struct S	0
		{	{	0
		<type> <name>;	int i;	1
		…	char ch;	1
		} <struct_name>;	} <struct_name>;	2
		using namespace <name>	using namespace std;	1
		<type> <name> (< parameter_list>)	void main()	0
		{	{	0
		…	printf(“hello”);	1
		}	}	1

COMPILER DIRECTIVES

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
CDL1	directive types	#define <name> <value>	#define MAX_SIZE 100	1
		#include <library_name>	#include <stdio.h>	1