C/C++ CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
2/18/2016	1.0	Original Release	Matthew Swartz
3/9/2016	1.1	Added Complexity Rules	Madhvi Kansagara
7/8/2016	1.2	Updated complexity table	Matthew Swartz

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.

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 C/C++ keywords that denote data declaration lines:

long long int	long double	long long	namespace	protected	short int	abstract
char16_t	char32_t	char64_t	explicit	long int	operator	register
template	typename	uint16_t	uint32_t	uint64_t	unsigned	volatile
char8_t	int16_t	int32_t	int64_t	mutable	private	typedef
uint8_t	virtual	wchar_t	double	extern	friend	inline
int8_t	public	signed	size_t	static	string	struct
class	const	float	short	union	using	auto
bool	char	enum	long	void	asm	int

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 C/C++ keywords that denote compiler directives:

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

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. C/C++ comment delimiters are "//" and "/*". A whole comment line may span one line 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.

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, ? 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

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	"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	

3. Examples

EXECUTABLE LINES

SELECTION Statement

ESS1 – if, else if, else and nested if statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
if (<boolean expression>) <statements>;	if (x != 0) printf (“non-zero”);	1 1
if (<boolean expression>) <statement>; else <statement>;	if (x > 0) printf (“positive”); else printf (“negative”);	2 1

if (<boolean expression>)
  <statements>;
else if (<boolean expression>)
  <statements>;.
.
.
else <statements>;
	if (x == 0)
	  printf (“zero”);
	else if (x > 0)
	  printf (“positive”);
	else
	  printf (“negative”);
		1
		1
		1
		1
		0
		1

if (<boolean expression>)
{
  <statements>;
}
else
{
  <statements>;
}
 	
NOTE: complexity is not considered, i.e. multiple “&&” or “||” as part of the expression.
	if ((x != 0) && (x > 0))
	  printf (“%d”, x);
	
	if (x != 0)
	{
	  printf (“non-zero”);
	}
	else
	{
	  printf (“zero”);
	}
		1
		1
		
		1
		0
		1
		0
		0
		0
		1
		0

ESS2 - ? : operator

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
Exp1?Exp2:Exp3	x > 0 ? printf (“+”) : printf (“-”);	1

ESS3 – switch and nested switch statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

switch (<expression>)
{
  case <constant 1> :
  <statements>;
  break;
  case <constant 2> :
  <statements>;
  break;
  case <constant 3> :
  <statements>;
  break;
  default
  <statements>;
}
	switch (number)
	{
	  case 1:
	  case 11:
	  foo1();
	  break;
	  case 2:
	  foo2();
	  break;
	  case 3:
	  foo3();
	  break;
	  default
	  printf (“invalid case”);
	}
		1
		0
		0
		0
		1
		1
		0
		1
		1
		0
		1
		1
		0
		1
		0

ESS4 – try-catch block

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

try
{
  // code that could throw
  // an exception
}
catch (exception-declaration)
{
  // code that executes when
  // exception-declaration is thrown
  // in the try block
}
	try
	{
	  cout << "Calling func \n";
	  Exception e)
	{
	  cout << “Error: “ << e;
	}
		1
		0
		1
		1
		0
		1
		0
		1
		0

ITERATION Statement

EIS1 – for loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

for (initialization; condition; increment)
statement;
 	
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++)
	for (i = 0; i < 10; i++)
	  printf (“%d”, i);
	
	for (i = 0; i < 10; i++)
	{
	  printf (“%d”, i);
	}
		1
		1
		
		1
		0
		1
		0

EIS2 – empty statement (could be used for time delays)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for (i = 0; i < SOME_VALUE; i++) ;	for (i = 0; i < 10; i++) ;	2

EIS3 – while loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

while (<boolean expression>)
<statement>;
	while (i < 10)
	{
	printf ("%d", i);
	i++;
	}
		1
		0
		1
		1
		0

EIS4 – do-while loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

do
{
<statements>;
} while (<boolean expression>);
	do
	{
	ch = getchar();
	} while (ch != '\n');
		0
		0
		1
		1

JUMP Statement

EJS1 – return statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
return expression	If (i=0) return;	2

EJS2 – goto, label statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

goto label;	loop1:
	x++;
	if (x < y) goto loop1;
		0
		1
		2

EJS3 – break statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
break;	if (i > 10) break;	2

EJS4 – exit function

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
void exit (int return_code);	if (x < 0) exit (1);	2

EJS5 – continue statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

continue;	while (!done)
	{
	ch = getchar();
	if (char == '\n')
	{
	done = true;
	continue;
	}
	}
		1
		0
		1
		1
		0
		1
		1
		0
		0

EXPRESSION Statement		
ESS1 – function call		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<function_name> (<parameters>);	read_file (name);	1
ESS2 – assignment statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

<name> = <value>;	x = y;
	char name[6] = "file1";
	a = 1; b = 2; c = 3;
		1
		1
		3

ESS3 – empty statement (is counted as it is considered to be a placeholder for something to call attention)		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more ";" in succession	; ;;;	1 3
BLOCK Statement		
EBS1 – block=related statements treated as a unit		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

/* start of block */
{
<definitions>
<statement>
}
/* end of block */
	/* start of block */
	{
	i = 0;
	printf ("%d", i);
	}
	/* end of block */
		0
		0
		1
		1
		1
		0

DECLARATION OR DATA LINES

DDL1 – function prototype, variable declaration, struct declaration, typedef

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<type> <name> (< parameter_list>);	void foo (int param);	1
<type> <name>;	double amount, price; int index;	1 1

struct <name>
{
<type> <name>;
<type> <name>;
}
	struct S
	{
	int x;
	int y;
	};
		0
		0
		1
		1
		1

struct { <type> <name>; <type> <name>; } <name>;	struct { int x; int y; } S;	0 0 1 1 2
typedef <type> <name>;	typedef int MY_INT;	1
typedef struct <name> { <type> <name>; … } <struct_name>;	typedef struct S { int i; char ch; } <struct_name>;	0 0 1 1 2
using namespace <name>	using namespace std;	1

<type> <name> (< parameter_list>)
{
…
}
	void main()
	{
	printf(“hello”);
	}
		0
		0
		1
		1

COMPILER DIRECTIVES

CDL1 – directive types

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
#define <name> <value> #include <library_name>	#define MAX_SIZE 100 #include <stdio.h>	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
nexttoward	atan2	log10	++	else if	==	#dictionary	=	->
nearbyint	acosh	ilogb	--	switch	!=	#include		
remainder	asinh	log1p	>>	while	<=	#define		
nextafter	atanh	log2	<<	case	>=	#ifndef		
copysign	acos	logb	%	else	&&	#import		
scalbln	asin	log	^	for	||	#module		
llround	atan		+	if	!	#pragma		
scalbn	cosh		-		<	#endif		
tgamma	sinh		*		>	#error		
lgamma	tanh		/			#ifdef		
lround	cos					#undef		
llrint	sin					#using		
remquo	tan					#elif		
frexp						#else		
ldexp						#line		
expm1						#if		
hypot								
floor								
trunc								
round								
lrint								
modf								
exp2								
sqrt								
cbrt								
erfc								
ceil								
fmod								
rint								
fdim								
fmax								

fmin								
fabs								
exp								
pow								
erf								
nan								
abs								
fma								