IPOL Software Guidelines — version 1.02

May 20, 2015

IPOL reviews, uses, publishes and distributes some software provided by the authors. With the requirements and recommendations expressed in these guidelines, we intend to facilitate the production and review of verifiable and usable software for reproducible research.

Contents

In Brief: Check List, Check Service and Examples

The list hereafter is a summary of the guidelines, to quickly check a program. Some are requirements, others are only recommendations. This document details and explains the guidelines later.

- zip or tar/gzip archive `name_version.{zip,tar.gz,tgz}`, less than 2 MB
- everything into a `name_version/` folder
- file names with `a-z`,`A-Z`,`0-9`,`-`,`_`,`.`
- no hidden file, backup or useless file, no binary
- C89, C99 or C++98 code tested with `gcc -std=xxx -Wall -Wextra -Werror`
- portable code, 32/64-bits, nothing specific to an operating system
- only `libtiff`, `libjpeg`, `libpng`, `zlib`, `fftw`, `libgsl`, `libeigen`, `cblas` and `clapack` external libraries
- compilation with `make` or `cmake`, only standard options, `make` uses `$(CC)` or `$(CXX)`
- command-line non-interactive interface
- max 1 GB memory, max 30 s computation in the demo environment
- can read/write in PNG, TIFF, PNM, EPS, SVG, VRML or PLY format
- copyright attribution and GPL/BSD license info in every source file
- patent warning if needed
- `README.txt` essential information
- correct, clean code in English
- max 80 characters per line, max 1000 lines per file
- `main()`, algorithmic and auxiliary code in different files
- detailed comments for every function and every implementation step
- example input data and result

A service to check an program against some of these guidelines is available with examples of programs following the guidelines at `https://tools.ipol.im/swg_check/`. Authors can use this service to verify their code before submission, and by reviewers as a preliminary validation of the software[1].

[1] The absence of error reported by this service doesn't imply that all the guidelines are correctly followed. Some guidelines need a human review.

About this Document

Status

This document is the official IPOL Software Guidelines, version 1.02b, published on May 29th, 2014. It is immediately applicable and obsoletes previous versions. The reference version is available at `https://tools.ipol.im/wiki/ref/software_guidelines/`.

Revisions

When needed, future versions of this document will be published and will replace the current version. The current version will still be available, with a summary of the differences. This revision will be announced on the IPOL website[2] and the IPOL discussion list[3].

Changes from version 1.01:

- Add `libgsl`, `libtiff 4.x` and `libeigen 3.x` to the list of allowed external libraries.
- Expanded on the possibility to use other libraries, distributed with the published software.

Vocabulary

These guidelines are designed for a research journal whose articles include some software material. They are written for IPOL, but may be used by others.

In this document, the term "published software" designates the reference software implementation of an algorithm submitted for publication in a journal article. The "web demo" is an online facility used to test this software. An article **may** contain more than one software, and a demo **may** use more than one software.

In this document, the words **must**, **must not**, **should**, **should not**, **recommended**, and **may** are used to express required, recommended, and optional items. IETF RFC2119[4] describes their interpretation, and Annex A adds details in the context of these guidelines.

[2] `http://www.ipol.im/`
[3] `http://tools.ipol.im/mailman/listinfo/discuss`
[4] `http://tools.ietf.org/html/rfc2119`

License

Guidelines

1. Packaging and Content

1.1. Compressed Archive

A published software **must** be packaged as a compressed archive file. This file archive can either be a single volume .ZIP compressed archive or a GZIP compressed tar archive[6]. The size of the compressed archive file **should** be less than 2 MB. In the remainder of this document, we will use the terms "zip archive" and "tar/gzip archive" for convenience.

Annex B of this document provides some examples of programs that can be used to produce such compressed archives.

1.2. Archive Name, Program Name and Version

The compressed archive file of a published software **must** be named according to the `name_version.extension` pattern, where:

- `name` and `version` **must** consist only of lower case letters (`a-z`), digits (`0-9`), minus (`-`) and period (`.`) signs;
- `name` **must** be at least two characters long and start with a letter; it **must** indicate the name of the program; this name can be the name of the executable program file, or another name, at the author's will;
- `version` **must** start with a digit; it **must** indicate a version number for the program, in the sense that two different releases of the program **must** have two different version numbers; if no version numbering scheme is established for the program, the `YYYYMMDD` pattern based on the year, month and day of the release date **may** be used;
- `extension` **must** be `zip` for zip archives and `tar.gz` or `tgz` for tar/gzip archives.

[5] `http://creativecommons.org/licenses/by/3.0/`

[6] These file formats are defined by the PKZIP APPNOTE specification[7], version 6.3.2, for the .ZIP compressed archive format, the IETF RFC1952[8] for the GZIP compressed format, and the POSIX.1 ustar definition[9] for the tar archive format.

1.3. File and Folder Names

All the files and folders extracted from the compressed archive **must** be located inside a base folder named `name_version`, where `name` and `version` are identical to those used for the compressed archive file name. Absolute path **must not** be used for files and folders extracted from the archive.

The name of all files and folders composing the published software **must** consist only of lower or upper case letters (`a-z`, `A-Z`), digits (`0-9`), minus (`-`), underscore (_) and period (`.`) signs. They **should** start with a letter.

The names **should** provide a meaningful hint of the content of these files and folders.

1.4. Hidden and Useless Files

A published software **should not** include hidden files or folders or by-products of the tools used by the authors, such as (but not limited to):

- files inserted by file managers (`.DS_Store`, `.directory`);
- folders inserted by version control managers (`.svn`, `.git`);
- backup versions (`filename`∼, `filename.bak`).

The software **should not** be distributed with files not useful to build, use or study the implementation of the algorithm published in the journal.

2. Implementation

2.1. Source Code

A published software **must** include all the material necessary to build one or more executable program files implementing the algorithm published in the journal. This material **must** be provided in human-readable source code form. A published software **must not** be distributed with binary precompiled files if these files can be obtained from source code[10].

Annex C provides some information for the authors to help them perform some frequent implementation tasks.

[10] If the authors want to distribute binary versions, they can do it in the journal as supplementary material but not via the compressed archive of the published software covered by these guidelines.

2.2. Programming Language

The source code of a published software **must** follow the published standard syntax of one or more compiled programming languages. Only C89, C99, and C++98[11] can currently be processed. If the authors want to publish their program with another well-known and standardized compiled language (such as Fortran 90 or C11), they should contact the editorial board to investigate the possibilities.

Authors **should** test their C and C++ software with the `gcc` compiler in strict compilation mode[15] before submitting it to the journal. This code **should** compile without any warning from the compiler version available at publication time.

The source code **may** use the OpenMP 3.0[16] API for shared multiprocessing programming (parallel programming) but it **must** also compile and provide the same results (albeit slower) without OpenMP. This implies that every call to the OpenMP run-time library **must** be hidden behind `#ifdef _OPENMP` macros. Usage of OpenMP **must not** be tied to a specific number of processors and **must** only rely on the OpenMP standard, not on any vendor implementation.

Other techniques for accelerated computing, like OpenCL[17] and OpenACC[18], are not supported by the journal. These techniques can be used in a published software, but they will not be reviewed nor used in the web demo.

2.3. Portability

The source code of a published software **must not** require any extension of the language or its standard library, or any resource specific to a hardware environment, operating system or compiler. These extensions and resources **may** be used to achieve better performances if they are available but their availability **must** be detected during the compilation or execution and an alternative portable implementation **must** be used in their absence. This includes (but is not limited to)

- language dialects specific to a compiler (GNU C, Microsoft C);
- standard library functions specific to an implementation (`drand48()`);
- assembler code or "intrinsic functions" mapped to a processor instruction (`mulps`, `__builtin_ia32_mulps()`);

[11] C89 is defined by the ANSI X3.159-1989 Programming Language C standard[12], C99 is defined by the ISO/IEC 9899:1999 Programming languages — C standard[13], and C++98 is defined by the ISO/IEC 14882:1998 Programming languages — C++ standard[14].

[15] C89, C99 and C++98 code can be tested with `gcc -std=xxx -Wall -Wextra -Werror` where xxx is `c89`, `c99` or `c++98`.

[16] `http://www.openmp.org/mp-documents/spec30.pdf`

[17] `http://www.khronos.org/registry/cl/`

[18] `http://www.openacc-standard.org/`

- computing on specialized hardware (GPGPU);
- operating system calls (Win32 `GetSystemTime()`, POSIX `gettimeofday()`);
- file system locations (`C:\Documents and Settings`, `/tmp`);
- code specific to a memory model environment (32-bits, 64-bits).

Special attention will be given to the Linux 64-bit environment because it is currently the primary environment for the web demos. But the program **must** also be usable in other environments and this portability **must not** be limited to the Win32/POSIX alternative.

2.4. Compilation

A published software **must** be compiled by an automated non-interactive build procedure with `make` or `cmake`. This build tool **must not** be configured to use any special compiler.

The default build procedure **must** use standard compiler options only[19] : `-c`, `-D`, `-E`, `-I`, `-L`, `-l`, `-O`, `-o` and `-U`.

In addition, `make` **must not** call `gcc` or `g++` but **must** call the standard `cc` and `c++` aliases instead. To select the compilers at compile time, `make` **should** use the standard `$(CC)` and `$(CXX)` variables. When the language dialect matters (C89 vs. C99) , `make` **should** use additional variables:

```
C89=cc    # C98 compiler
C99=c99   # C99 compiler
CC=$(C99) # the default POSIX C compiler is c99

# compiling C89 code
foo.o : foo.c
        $(C89) -c foo.c -o foo.o
# compiling C99 code
bar.o : bar.c
        $(C99) -c bar.c -o bar.o
```

2.5. External Libraries

A published software **must** not use external software components except for the libraries and APIs listed hereafter. The program **may** expect these basic software components to be correctly installed and configured during the compilation and execution:

[19]Standard compiler options are defined by the POSIX c99 specification[20]. Other options depending on the environment (hardware, operating system, compiler) **may** also be provided, for example for an optimized compilation, but they **must not** be used in the default build procedure.

- `libtiff`[21] version 3.x or 4.x and `libpng`[22] version 1.4.x to read and write files, with their dependencies `libjpeg`[23] version 8.x, `zlib`[24] version 1.2.x;
- `fftw`[25] version 3.x (single and double precision) for Fourier transforms;
- `libgsl`[26] version 1.14+ for mathematical routines.
- `libeigen`[27] version 3.x for linear algebra[28].
- `cblas`[29] and `clapack`[30] for linear algebra.

Other libraries can be examined and may be added to this list on request, if they are portable, widely used, with a stable API.

This restriction only applies to software components used by the published software but not distributed in source form with the program. Other libraries can be used if provided with the published software, as detailed in the next section.

2.6. Support Code

Authors **may** include some code from other software projects and libraries in their published software to reuse standard and well-defined algorithmic blocks, if it helps improve the quality of the implementation. This support code **must** follow the current implementation guidelines: distributed in source form, written in a standard programming language, portable, automatically compiled, without external dependencies except for the ones allowed.

To be reusable, the published software **should** only use the support code via calls to algorithmic functions, and **should not** be based on it. The function calls to this code **should** be clearly identified, with a description of their expected behavior.

This support code **should** be kept as small as possible; a huge library **should not** be involved for a tiny algorithm: to compute the eigenvalues of a 2×2 matrix, authors **should not** use a linear algebra library.

[21] `http://www.remotesensing.org/libtiff/`
[22] `http://libpng.org/pub/png/libpng.html`
[23] `http://www.ijg.org/`
[24] `http://zlib.net/`
[25] `http://www.fftw.org/`
[26] `http://www.gnu.org/software/gsl/`
[27] `http://eigen.tuxfamily.org/`
[28] Only the "official" Eigen modules are allowed, no code form the "unsupported" experimental section.
[29] `http://www.netlib.org/blas/`
[30] `http://www.netlib.org/lapack/`

2.7. Usage and Input/Output

A published software **should** be minimal and only perform the algorithm published in the journal. It **must** be usable from the command line environment without any user interaction, taking all its parameters from the command line.

A published software **must** be able to read the input data and write the final output data in at least one of these formats[31]: PNG, TIFF or PNM for raster images, EPS or SVG for vector images, VRML or PLY for meshes, and plain text for other data. Other file formats can be added to this list on request, if they are clearly defined and widely used.

Annex C provides some information for the authors to help them use external libraries to read and write images.

A published software **must** return a success exit code (`exit(EXIT_SUCCESS)`) when the algorithm could be successfully executed, and a failure code (`exit(EXIT_FAILURE)`) it could not, for any reason such as (but not limited to) wrong command-line parameters, missing input files or wrong input data. Memory errors (segmentation faults) **should not** occur during the execution, and **must not** occur as the result of wrong command-line parameters.

2.8. Computing Resources

In the demo environment, a published software **should not** use more than 1 GB of memory and **must not** use more than 8 MB of stack memory space (for recursion, local variables and variable-length arrays). The program **must not** need more than 30 seconds to process typical data. For slow algorithms, this limit **may** be achieved with parallel processing or a limit on input size.

Annex C provides some information for the authors to help them improve the performance of their implementation and check the memory used.

3. Copyright, License and Patents

3.1. Copyright Attribution

Every source code file in a published software **must** mention its authors in a copyright attribution line at the top of the file. This mention **may** be omitted in trivial files such as

[31] PNG is defined by the IETF RFC2083[32], TIFF is defined by the Adobe TIFF 6.0 Specification[33], PNM (PBM, PGM and PPM) is defined by the netpbm documentation[34], EPS is defined by the Adobe Encapsulated PostScript 3.0 Specification[35], SVG is defined by the W3C Scalable Vector Graphics 1.0 Specification[36], and VRML is defined by the ISO/IEC Virtual Reality Modeling Language Specification[37]. There is no formal published specification of PLY, but this simple format introduced by the Standford 3D Scanning Repository[38] is documented on Paul Bourke's site[39]. Plain text output **should** be understandable by a human reader and easy to parse with a software.

header code.

Every person whose contribution to this file is not trivial and implies some creative work **must** be credited. Of course, if the authors use or modify a file previously written by other persons, the copyright attribution to the previous authors **must not** be removed. The copyright attribution **must** include the years of production of the work, the full name and an e-mail address for contributor. It **may** also include other relevant information such as the employer, affiliation or web site.

An simple example for a single author can be:

```
Copyright (C) 2011, Jane Doe <jane.doe@example.org>
```

A complex example with three contributors can be:

```
Copyright (C) 1998-2003, Taro Yamada <taro.yamada@example.jp>
Copyright (C) 2005-2011, Juan Perez <juan.perez@example.es>
Copyright (C) 2011,      Marie Untel, ENS Cachan
                         <marie.untel@ens-cachan.fr>
```

3.2. Patent Warning

When the authors are aware or suspect that a source code file implements an algorithm which might be linked to a patent (the main algorithm published in the journal or another algorithm used for this implementation), a patent warning **must** be inserted after the copyright attribution, in every file potentially linked to this patent. This wording is **recommended**:

```
This file implements an algorithm possibly linked to the patent
<REFERENCE OF THE PATENT>.
This file is made available for the exclusive aim of serving as
scientific tool to verify the soundness and completeness of the
algorithm description. Compilation, execution and redistribution
of this file may violate patents rights in certain countries.
The situation being different for every country and changing
over time, it is your responsibility to determine which patent
rights restrictions apply to you before you compile, use,
modify, or redistribute this file. A patent lawyer is qualified
to make this determination.
If and only if they don't conflict with any patent terms, you
can benefit from the following license terms attached to this
file.
```

3.3. License

Every source code file **must** mention a usage and redistribution license after the copyright attribution (and patent warning for algorithms potentially linked to a patent). Of course, if the authors use or modify a file previously written by other persons, the license chosen by the previous authors **must not** be modified.

- When the authors are not aware of a possible patent issue, the license **must** be one of the following free software licenses: GPL[40], LGPL[41], AGPL[42], BSD[43], or the CC0[44] dedication.
- When a source code file can be linked to a patented algorithm and the source code authors are not the patent inventors, the file **must** be distributed under the BSD license or CC0 dedication.
- When a source code file can be linked to a patented algorithm and the authors of the source code are the patent inventors, the file **must** be distributed either under the BSD license, the CC0 dedication or "for research and education only"[45].

These wordings are **recommended**:

```
This program is free software: you can use, modify and/or
redistribute it under the terms of the GNU General Public
License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later
version. You should have received a copy of this license along
this program. If not, see <http://www.gnu.org/licenses/>.

This program is free software: you can use, modify and/or
redistribute it under the terms of the simplified BSD
License. You should have received a copy of this license along
this program. If not, see
<http://www.opensource.org/licenses/bsd-license.html>.

To the extent possible under law, the authors have dedicated all
```

[40] http://www.gnu.org/licenses/gpl.html
[41] http://www.gnu.org/licenses/lgpl.html
[42] http://www.gnu.org/licenses/agpl.html
[43] http://www.opensource.org/licenses/bsd-license.php
[44] http://creativecommons.org/publicdomain/zero/1.0/
[45] Distribution "for research and education" can help avoid conflicts between patent rights and software license when the patent inventors are the source code authors. It is not needed in other situations because the validity of the license will depend on the local patent regulations, as stated in the last sentence of the patent warning.

```
copyright and related and neighboring rights to this software to
the public domain worldwide. This software is distributed without
any warranty. You should have received a copy of the CC0 Public
Domain Dedication along with this software. If not, see
<http://creativecommons.org/publicdomain/zero/1.0/>.

This program is provided for research and education only: you can
use and/or modify it for these purposes, but you are not allowed
to redistribute this work or derivative works in source or
executable form. A license must be obtained from the patent right
holders for any other use.
```

The exact terms can differ, for example when a different GPL/LGPL/AGPL license version is chosen. The full text of the license **must** be included in a separate file with the source code.

Authors **should** verify that the usage of these licenses for their software publications complies with their employer policy and local situation and jurisdiction.

4. Documentation

4.1. README.txt

Every published software **must** provide a file named `README.txt` in the base folder and written in plain text and in English. This `README.txt` file **must** include the following essential information, in any order:

- name and brief description of the program
- reference to the article
- authors and contact information
- version number and release date
- location of future releases and updates
- copyright, patent and license information
- tools and libraries needed to compile and use the program
- compilation instructions
- usage instructions and example
- changes in the software since it was first published

- list of the files, mentioning which files are reviewed
- list of known defects
- credits and acknowledgments

If this information is too long, the `README.txt` file **may** be split: compilation instructions in `INSTALL.txt`, usage instructions and example in `USAGE.txt`, changes in the software in `CHANGES.txt` and list of files in `MANIFEST.txt`. These secondary information files **must** be referenced in `README.txt`.

The `README.txt` file **may** contain other information, and **may** also be completed by another documentation, possibly with more details, in text, PDF, HTML or any other format.

For a simple code, the license information in `README.txt` can be

```
This program is written by Jane Doe <jane.doe@example.org> and
distributed under the terms of the GPLv3 license.
```

A complex case (multiple authors, patents and licenses) can be:

```
This program is written by Taro Yamada <taro.yamada@example.jp>
and Juan Perez <juan.perez@example.es> with contributions from
Marie Untel, ENS Cachan <marie.untel@ens-cachan.fr>.
- mmatch.c and rot_tree.c may be linked to the pending EU patent
  123.456 by Taro Yamada and Juan Perez and are provided for
  scientific and education only.
- demoz.c may be linked to the US patent 65.43.21 by Jane Doe;
  see the file for license terms.
- eizo.c and linalg_lib.c are distributed under the terms
  of the BSD license.
- All the other files are distributed under the terms of the
  LGPLv3 license.
```

4.2. Readability

In a published software, the source code is a primary material for the publication. It will be reviewed, published and read like any other part of the article. The authors **must** take care of the clarity of their program.

The source code of a published software **must** be consistently indented and spaced. Lines **should** be limited to 80 characters and **should not** end with blank characters (spaces, tabs, ...). Files **should not** have more than 1000 lines. The line terminations **should** be the same (DOS/Windows `CRLF` or UNIX `CR` style) for all the files of the program.

Functions **should** be grouped by abstraction level in different source code files:

- the `main()` function, command-line processing and input/output calls in one file,
- the implementation of the algorithm described and reviewed in the article in one or more other files,
- and the implementation of auxiliary and external routines in one or more other files.

Annex D provides some examples of programs that can be used to improve the indentation, spacing and presentation of a source code.

4.3. Implementation and Comments

The source code of a published software **must** be commented precisely and exhaustively. Authors **should** target the "1/8 comment/instruction ratio", but the quality of the comments is more important than the quantity. The source code **must** be written in English, including all variables, functions names and comments.

Authors **must** ensure that the code is understandable, to the satisfaction of the editor and reviewers, so that consistency between the description of the algorithm and its implementation can be verified. The relation between each part of the implementation and the respective part of the description of the algorithm **must** be explained in the comments.

Authors **should** apply simpler implementations when available, follow the conventions of the programming language, and use comments to explain implementation choices and every complicated or subtle point in the program. Clarity is more important than virtuosity.

Every function **must** be documented with at least one line explaining what the function is doing, and the meaning of its parameters and return value.

The Doxygen[46] source code documentation format is **recommended** for every published software.

Annex D provides some examples of programs that can be used to count the comment, instruction and blank lines.

4.4. Example Data

The authors **should** provide an example of input file to test the published software and the result to expect when this input file is processed by the program.

[46] `http://www.stack.nl/~dimitri/doxygen/`

Annexes

A. Key Words

The key words **must**, **must not**, **required**, **shall**, **shall not**, **should**, **should not**, **recommended**, **may**, and **optional** in this document are to be interpreted as described in IETF RFC2119[47].

must This word, or the terms **required** or **shall**, means that the definition is an absolute requirement.

must not This phrase, or the phrase **shall not**, means that the definition is an absolute prohibition.

No article will be published if a program included in this article doesn't follow such requirements or prohibitions.

should This word, or the adjective **recommended**, means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

should not This phrase, or the phrase **not recommended**, means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

The rationale for not following such recommendations **must** be agreed by the authors, reviewers and editor before an article is accepted for publication.

may This word, or the adjective **optional**, means that an item is truly optional. One author may choose to include the item because a particular article requires it or because the author feels that it enhances the software while another author may omit the same item.

B. Compression and Archive Tools

Authors can consider that the files produced by the following tools are correct zip and tar/gzip archives:

[47] `http://tools.ietf.org/html/rfc2119`

- on Linux systems, the `tar` (usually "GNU tar"), `zip` (usually "Info-Zip zip") and `gzip` programs;
- on Mac OS X systems, the "Create Archive" feature of the graphical interface and the `tar` (usually "BSD tar") and `gzip` programs;
- on Windows systems, the "Compressed Folder" feature of the graphical interface and the `7-zip`[48] program.

C. Coding Help

The authors of the article do not need to be the authors of all the implementation of their algorithm. The original copyrights and licenses of the reused code parts must of course be respected. This external code will not be reviewed, and it can not be updated once the article is published.

Some help to read and write image files is available[49] with simplified interfaces to `libpng` and `libtiff` and some examples. Authors can also find a list of contributions and tools[50] from other authors willing to share.

You can set limits on computational resources with `ulimit -m 1000000 -d 1000000 -v 1000000 -s 8000 -t 30` before you run the program (in a `bash` console, here with values for for 1 GB memory, 8 MB stack, and 30 s computation time). Memory allocation beyond these limits will be refused and the program will fail.

The editorial board can provide some help to the authors to accelerate their code and guide them for performance profiling and parallel programming. The discussion list[51] is the good place for any questions related to the journal, including the implementation work.

D. Source Code Tools

Authors can use the `expand`[52] program (Linux, Mac OS X) to convert tabs to spaces. They can improve the presentation and clarity of their software with `indent`[53] (C, Linux), `uncrustify`[54] (C/C++, Linux and Windows), `astyle`[55] (C/C++, Linux, Mac OS X and Windows) and `UniversalIndentGUI`[56] (cross-platform graphical front end).

[48] `http://www.7-zip.org/`
[49] `https://tools.ipol.im/wiki/doc/tools/`
[50] `http://tools.ipol.im/wiki/author/code/hatchery/`
[51] `http://tools.ipol.im/mailman/listinfo/discuss`
[52] `http://www.gnu.org/software/coreutils/manual/html_node/expand-invocation.html`
[53] `http://www.gnu.org/software/indent/`
[54] `http://uncrustify.sourceforge.net/`
[55] `http://astyle.sourceforge.net/`
[56] `http://universalindent.sourceforge.net/`

Tools like `cloc`[57], `ohcount`[58] and `sloccount`[59] can be used to count the comments, instructions and blank lines and evaluate the comment/instruction ratio.

[57] `http://cloc.sourceforge.net/`
[58] `http://ohcount.sourceforge.net/`
[59] `http://sloccount.sourceforge.net/`