How do you ensure that you as programmer have written quality C code? [closed]

Asked 15 years, 11 months ago Modified 12 years, 10 months ago

Viewed 2k times

4

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 9 months ago.

Improve this question

I m looking to write some quality C code. Can someone point me to some articles , websites..whatever I need something with examples. I have already seen and read K&R C book.

But times have changed, some one must have more to say on quality C Code. and another important thing is How do you ensure that you as programmer have written quality C code??

c software-quality

Share

Improve this question

Follow

edited Jan 20, 2009 at 19:53

Wouter van Nifterick

24.1k ● 7 ● 81 ● 123

asked Jan 11, 2009 at 12:32

linkedlist

307 ● 4 ● 8

A previous discussion, what-are-some-good-resources-for-learning-c-beyond-kr, might point to more (books) examples. – gimel Jan 11, 2009 at 12:52

9 Answers

Sorted by: Highest score (default)

12

Someone mentioned some compiler switches, but having syntactically smooth code is not going to ensure a quality end-product, because there's more to software quality than that.

There are several classifications of software qualities, but here's a list that you can use as a checklist:

- **Correctness** (does it work according to spec?)
- **Reliability** (can de user depend on it?)
- **Robustness** (does it work in unexpected situations?)

- **Performance** (does it do the job fast enough for the user?)
- **Usability** (is it user-friendly?)
- **Verifiability** (can its properties be verfified easily?)
- **Maintainability** (can modifications be made easily?)
    - **Repairability** (can defects be fixed within reasonable time?)
    - **Evolvability** (can new functionality be added simply?)
- **Reusability** (can the code easily be used in other projects?)
- **Portability** (can it run easily in different environments?)
- **Understandability** (can maintainers easily understand it?)
- **Interoperatability** (how well does it cooperate?)
- **Productivity** (efficienct and performant delivery)
- **Timeliness** (ability to deliver on time)
- **Visibility** (are all steps documented clearly?)

edited Jan 11, 2009 at 13:14

Jonathan Leffler

752k ● 145 ● 946 ● 1.3k

answered Jan 11, 2009 at 13:01

Wouter van Nifterick

24.1k ● 7 ● 81 ● 123

2 he specifically asked for code quality, not software quality; but +1 nonetheless - many of the points are still valid... – Christoph Jan 11, 2009 at 13:09

1 Nice list. I'd add, **Cost** (other things being equal, cheaper is better) – ChrisW Jan 11, 2009 at 14:25

Code review and coding conventions definitely missing from this list, but sure it's much better than just listing compilation flags, that can help you achieve part of the goals above, but can't be defined as way to get good code. – Ilya Jan 11, 2009 at 16:58

BTW all this relevant to any language not only in C, so writing good code is same in all languages techniques can be different but high level definitions are same .. – Ilya Jan 11, 2009 at 17:00

10

Enable warnings in your compiler. With gcc, I use these flags:

```
-std=c99 -pedantic -Wall -Wextra -Werror -Wmissing-pro
-Wmissing-declarations -Wshadow -Wpointer-arith -Wcast
-Wredundant-decls -Wnested-externs -Winline -Wno-long-
-Wstrict-prototypes
```

If your code can't be changed to not produce warnings, drop the `-Werror` or don't use the specific flag producing the warning.

Share Improve this answer

Follow

answered Jan 11, 2009 at 12:41

Christoph

169k ● 36 ● 186 ● 241

2 This can lead to syntactically correct code, not quality-wise code. – friol Jan 11, 2009 at 13:17

@friol: correct code is a prerequisite for quality code; also, this will emit warnings for a lot of things which are no longer considered best practice (implicit type conversion, improper prototypes for functions, use of non-standard language features,...) – Christoph Jan 11, 2009 at 13:25

1 +1: Though I agree with friol, it's worth allowing the compiler to do the grunt work, and to remove as many distractions as possible. – Adam Liss Jan 11, 2009 at 14:24

@friol, what a shallow thought… – poige Apr 28, 2012 at 10:42

Traditionally, people have used lint to help out with this.

7

Share Improve this answer

Follow

answered Jan 11, 2009 at 12:43

Dave Markle

97.6k ● 20 ● 151 ● 171

Use a static analysis tool, traditionally called lint, however I've used splint which is good. See recommendations in this question. Personally I'd recommend enabling warnings and fixing them.

5

In terms of the rules

- Do not trust input data - validate everything, size, type, content.
- Protect against buffer overruns - strcat and many others aren't safe.
- Do use unit testing,ddj article.
- Do get your code reviewed by someone else
- Do not make assumptions.
- Do keep functions short, and test each one thoroughly.
- Use meaningful names.
- Write readable code.
- Don't be lazy - if you need to change a something to make it more meaningful then do it sooner rather than later.

Edit: Specific to C, this list of C gotchas is essential reading, and even though it is for C++ it is worth going through the CERT C++ Secure Coding Standard

Share Improve this answer

Follow

edited May 23, 2017 at 12:10

Community Bot

1 ● 1

answered Jan 11, 2009 at 13:18

Richard Harrison

19.4k ● 4 ● 42 ● 67

" validate everything, size, type," But type checking is often considered to be of poor quality and a code smell. I'm not

sure if you can even do type checking in C. – Mehdi Charife Sep 22, 2023 at 19:25

1 In the context of not trusting input data what I meant was to validate the input data (from the user or external system) prior to processing. So if you had a string field that you will convert into an integer ensure that it only contains numbers, no decimal point. If you are converting a string into a double then permit a decimal point. – Richard Harrison Mar 11 at 7:27

2

People have so far mentioned tools. However, beyond a certain point, there is really only one thing you can do to really improve the quality of the code you write:

Write code.

Share Improve this answer

Follow

answered Jan 11, 2009 at 13:14

user51568

2,633 ● 2 ● 20 ● 23

2

Write unit tests!

It might be a bit more cumbersome to write unit-tests in C compared to more modern languages, but it is still very much worth it.

I would say that proper unit tests are the #1 way to ensure the quality of any code. You can use all the static analysis tools and code reviews you want, but nothing beats actually running the code and verifying the results.

Share Improve this answer Follow

answered Jan 11, 2009 at 13:38

Rasmus Faber
49.6k ● 25 ● 148 ● 193

2

It depends in part on what you mean by 'quality C' code.

One important aspect of the program is "does it do what it was designed to do"? That is hard to measure, but is crucial.

Then you need to know whether the code is acceptable to compilers - using the set of GCC compiler options provided by Cristoph would indicate that the code is in good shape. (Although I would quibble over `-Wno-long-long`, that depends on where your code might need to be be moved to).

The code layout is important. Is the code readable by humans as well as compilers? Is the layout uniform? Is it in one of the standard formats - there are several, all with major followings, and as long as you use one of them consistently, the code should be fine. Is it appropriately commented? That means enough comments, but not too many! The file should have a header comment indicating what's in it - and probably who wrote it, and maybe the licence under which it is distributed. There have been multiple questions on SO about that, including Professional #include comments. Writing code to a good layout standard is routine after a short time, though.

Documentation may be relevant - usually is relevant. How would someone else know that the code exists, what it does, how to use it, when to use it, when not to use it?

The code should be written with good enough algorithms - it should not use exorbitant amounts of memory, disk, or CPU time. It should also not leak resources. There's also no point in wringing the last CPU cycle of performance enhancement out of a routine that will be used once per run of a program, for a few milliseconds, when it starts up, unless you can demonstrate that it is a performance bottleneck for the program as a whole.

Wouter van Nifterick has given an excellent set of pointers too.

Share Improve this answer Follow

edited May 23, 2017 at 10:28

Community Bot

1 ● 1

answered Jan 11, 2009 at 13:16

Jonathan Leffler

752k ● 145 ● 946 ● 1.3k

1

It's tough to see for your own self whether or not you are writing quality code, sure there's tons of automation and standards out there, but how can one apply everything they've seen out there?

This is where I'm a big fan of peer review as a method of judging code quality. Let other see (and also learn) from

your code and that'll be the judge of quality.

Share Improve this answer

Follow

answered Jan 11, 2009 at 13:06

tekiegreg

1,707 ● 6 ● 25 ● 42

> As with editing a manuscript: leave it alone for a week and look at it again with fresh eyes, you find that you can catch many more of you won mistakes...
> – dmckee --- ex-moderator kitten Jan 11, 2009 at 14:41

0

Some modern software lifecycle practices that enforce quality of code:

- integration tests (planned at project startup)
- peer reviews of code (during development)
- pair programming (during development)
- the use of consolidated libraries (instead of a 'reinventing the wheel' approach)

The latter one may in particular apply to the C language.

Share Improve this answer

Follow

answered Jan 11, 2009 at 13:16

friol

7,086 ● 4 ● 49 ● 82