

Re: Learning C

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-03/msg01657.html

- *From:* "Richard G. Riley" <rgrdev@xxxxxxxxxx>
 - *Date:* 11 Mar 2006 18:48:19 GMT
-

On 2006-03-11, Nick Keighley <nick_keighley_nospam@xxxxxxxxxxxxx> wrote:

Richard G. Riley wrote:

"Nick" posted the following on 2006-03-11:

Richard G. Riley wrote:

"Chris" posted the following on
2006-03-10:

Richard G. Riley wrote:

"Nick" posted
the
following
on
2006-03-10:

I
thought
you
were
implying
Linux
was
better
because
the
debugger
was

There is no
"the
debugger" :

Re: Learning C

although
gdb is
prevalent in
Linux –
albeit with
several
front ends.

"debugger" / "debuggers" whatever. I've used ddd.

ddd is a front end to gdb.

I know

better.
I've
never
stepped
through
an
existing
application
(that
wasn't
broken)
with
a
debugger.
If
you
say
it's
a
good
way
to
learn
C,
who
am
I
to
argue.

I stress that I talking about "stepping through an application".

Re: Learning C

I have
no
objection to examining existing code. One way to learn is to
look at
good
examples (and sometimes at bad).

And how can stepping through an app be bad? Were you never put onto a
new project with a code base of several hundred thousand lines of code
and told to isolate some relatively bugs to get you familiar with the
code base?

about a year ago I was put on 750 kloc application. I had no previous
experience with the application. And limited experience with the
programming language. I did not step through the code with a

debugger.

Your choice. I would have. Especially when I modified the code. But
again you are offering no evidence to suggest that stepping through an
app is "crazy" or "bizarre" as you originally claimed.

I said its
one way to
get used to
the structure
and flow of
applications
which is
what he
wants. Also,
I do think a
debugger
can
give real
insight into
how C
works in the
real world :
results of
operators
there for
you to see
with no
overhead of
printfs

Re: Learning C

which some
favor.

The printf's are portable.
The printf's work without
manual
intervention.

No they dont : you have to insert them in the
code.

yes, but you don't have to keep on inserting them. Debuggers
are
generally manual.

<snip>

I meant me pointing out they are useless in most server based apps or
message driven GUI apps.

there are alternatives to printf(). I usually use some sort of logger.
For both
servers and GUIs. That 750kloc application has both.

Fine. But often loggers affect the app. And again, you only see what
you are told you can see.. : a debugger lets you see what you want to
see when you want to see it. Being able to see the data greatly
simplifies understanding an application and I cant expect anyone to
dispute that. But someone will ... :-;

Its rare that I find someone wanting their
printfs to be
portable in a system process or a an X gui or
a Win 32 winproc : they
dont work. Home grown or system supplied
logging libraries possibly :
but can you really analyse them at run time?
I cant : I like to step
through and see the flow of the app to get a
feel for how the systems

Re: Learning C

heart is beating.

ok I'm not saying you are wrong. I'm just pointing out that not everyone works and learns the way you do. I don't single step debuggers to examine other people's applications

Thats fine Nick : I dont expect you too – but you seem to have strong reasons for making a point against it whereas I see *only* benefits.

ok, we disagree. What's wrong with *reading* the code. UML? Source browsers?

Because not all code is readable? Because not all code can be properly deciphered from reading? Because you dont know what data is coming up from those functions 40 deep? Becuase its plagued with misplaced casts?

You dont know what branch conditions are going to be taken based on HW port values? There are numerous reasons.

The printf's work without having to understand an additional tool.

But
it
still
sounds
bizzare
to
me

What does?

using a debugger to examine existing applications

Re: Learning C

We wont get into the pissing contest of who has worked on more apps/platforms etc but I find it a good teaching tool to get people up to speed with an app and its internal data structures : modify on the fly, symbol tables etc. Cant do that with a print out or printf's.

I have **never** had any desire to do these things. Find the bug. Fix the bug. Why would you want to modify things on-the-fly?!

Modify data to test or force a state. Its a basic of debugger use : you can force your hand to avoid having to wait for a certain random data suite to trigger the bug.

e.g someone says I'm getting a divide by zero somewhere near func().

go to func in debugger, force a zero into the divisor and see why the checks arent stopping it. Simple example.

Are we
talking
about the
same thing?
Do you
doubt that
watching
other, well
written apps
work is
beneficial to
a newbie?

examining existing applications and other examples, yes.
stepping existing applications and other examples, no.

for me "examining" is "stepping" : but obviously with some strategically placed break points and a few return to callers etc :)

"examine", to me, means to study the source code. A good source browser can be handy.

Re: Learning C

A good debugger is that too.

It
seems fairly
clear to me
that it can
only help.
Its how the
entire
Open SW
system
works :
people
learning by
doing and
picking up
on
other
peoples
work.

right. BUT NOT IN A DEBUGGER

Really : in a debugger. Do you really advocate printf's over a debugger
in a huge code base?

:~)

How do you map data values to their equivalent
constants? To me it sounds incredible. If you hadnt mentioned ddd I
would wonder if we are talking about the same thing.

some constants get dumped as numbers. Some enums are already wrapped
so as to be able to decode themselves (ok I admit this is C++, ...its
only a
little bit of a move to the Dark Side. I can stop any time I want).

I'm not good at remembering what 0xfecda means. I prefer something in
the debugger like "WM_GETFOCUS" or some such.

Re: Learning C

Re: Learning C

You've conflated "learning by doing and picking up on> other peoples work" with "watching ... apps work" and that with stepping through an application using a debugger. I think that's misleading.

Really? Seems pretty straightforward to me and also how close on 100% of Universities teach coding at some stage or other :

really? *all* universities encourage the use of stepping? Do you have statistics?

Maybe stepping is the key word here : it does not mean every line. It means strategically placed break points and data analysis at those points. It is an art.

and *all* universities teach it? Its a long time since I was at university.

me too.

Yorkshire Man 2:
"Symbolic debuggers! You were lucky we 'ad t'punch t' cards with ar teeth!"

:)

Could you go back and *read* what I and other posters have actually been saying?

I did. Some people seem to think that a debugger is just for finding

Re: Learning C

bugs : it is not. It is also useful for examining runtime trends. Far more useful than hard to decipher printf's, especially in a non console mode.

but you have to know where to put the breaks. And that means you must at least partially understand the application. To do that I'd read the code...

So do I: of course. But i also analyse the flow at the same time : a debugger does let you read the code you know :-;

adding modules to existing systems.

yes! absolutly!

Maybe I'm a bit slow today but I'm not seeing the subtleties of the point you are making here :

well you don't "add to an existing module" by using a debugger

Of course you do. You use it to examine the data flow between the system and your module.

well I add to an existing module with a Text Editor. I'm not being funny here we *really* seem to be missing each others point.

No you dont. You use a text editor to create the symbolic equivalent which is then compiled & linked into the bigger app. The debugger can then be used to force data between your interfaces and the legacy very easily for the purposes of quick integration testing. Yes, there are other ways too.

I REALLY don't use a debugger to add code to an existing module.

Re: Learning C

No. Nor do I. I use it, frequently, to monitor the addition. No one here is suggesting the debugger is a compiler & linker & editor. It is for me part of the process. Not for all, but we're heading back down that well covered track again :)

the user is looking for
a way to learn how to structure applications
and build them himself.

yes yes

I fail to see how analysing existing,
successful apps can be anything
other than beneficial. It doesnt take away the
donkey work of learning
the lanugage basics, but it can make text
book "science" much more
accessible and "real". I cant imagine
becoming a programmer without
such practice, guidance and "practical
training". Its the same in all
walks of life.

I think I'll give up here. My point wasn't that important. Just
trying
to
make the point that different people do thinks in different
ways. Linux

is not the only platform. Not everyone uses debuggers the
way you do

Nobody said it was : you came flying in with that.

riight

You did. Look at the thread. The OP mentioned Linux : I said it was a
good idea for what he wanted to do : you came in with "hold on hold on
lets not start an OS war here".

Re: Learning C

The OP had expressed an interest in Linux and the fact that it comes with free industry strength compilers and debuggers as well as hundreds of rock solid C apps complete with build packages makes it a good bet for someone to learn how to structure and analyse an application in C. Windows can not compete with that IMO.

it's a perfectly good development platform though. Look Linux is ok. Windows is ok. But two posters had encouraged the OP to move to Linux. I'm not sure it's such an open and shut decision. You can write crap non-portable code on Linux as well.

But you don't have the facilities to support his learning as well IMO : remember me mentioning the thousands of apps with source code? No one is saying Linux is "better", but I would say it's better for someone wanting to get into C programming big time : everything is part of the base install. For Free. And documented.

(some people only use them when they suspect a compiler error). A

If there is a compile error, you won't be using a debugger.

well compiler bugs are about as rare as hens teeth. But why not? The debugger would show you that the program flow or calculation results did not follow the source code. All my tools are broken if the source code does not represent what the machine is doing.

To be honest I can't remember the last time I had a bug caused by a compiler error

Different things. I meant compilation error : sorry.

newbie should be aware there are different ways to do things and not lock themselves into a particular approach too early.

A newbie needs to examine other people's stuff : especially when adding/fixing bugs. Approaches vary, but it can never hurt to get down

Re: Learning C

and dirty to be sure.

A newbie certainly shouldn't dedicate his life to printf()'s : they simply do not work in a lot of environments, are inflexible, and only show the data that you WANT to see, not the data you SHOULD see. This is where a debugger's "locals" view etc comes in. You see all local data. Not what someone thinks they need to see.

Maybe I'll even get my debugger out and step through a program sometime to see if it brings me any insights. Maybe you should try a "debugger free day" and try and see what you have to do to manage without. Try reasoning about the program. Consider invariants and post/pre conditions. Try adding asserts etc.

If you recall I did mention that a home brew logging system is preferable to printf :

where did the words "homebrew logging system" appear in the paragraph above? I did write a fairly blunt paragraph here. And then thought better of it.

Do you know what an invariant is? Design by contract?

I used those words. It's when you wrap whatever underlying logging system is convenient to you in a fairly generic calling interface : so you could log to files, text consoles, window systems whatever without changing the calling code.

I have nothing against them and have implemented many with various backend report generators to examine the data.

good!

A debugger isn't the only tool : and I never said it was. What it can do is allow you to see the flow of an application while watching live data, allow you to modify that data

Re: Learning C

never felt the need. Not since compilers got fast enough to run during the day.

I really dont understand this. What has that got to do with a debugger?

and to examine and ensure
data typing is consistant from skillful use of register/pointer/memory
examinations : it is why they exist.

you examine registers? On a deeply embedded system, ok. But a
server?

I also mentioned memory and pointers. And yes I do. Very useful in
debugging big C systems.

no way is the One True Way

And no where did I say it was. The whole crux here is you doubting
that stepping an existing app can help a user understand it : after
many, many and varied projects on various platforms in various
languages I find it incredulous how you could doubt this would be
beneficial. In order to even put in these printf() you need some
understanding of whats going : that does not come from a print out all
the time. It does not come from a func spec which is not always
there. It does not come from holistic overview : it comes from
stepping through and examining what is going on. From finding out when
and why certain modules are called. From knowing what that little bit
of bit fiddling results in, from forcing a function call at runtime in
order to test a different calling parameter. The list goes on. Of
course none of this is "the only way" : but its a good way and one
thats been used a lot for years.

I'd say the same went (in spades) for stepping. You've got to know
where
to step. There's source code, case tools, source browsers, source code.
I've even resorted to grep to find callers of functions.

Re: Learning C

grep is ok : if I dont have a decent IDE I use something like emacs tags. I have used grep in about 15 years :)

Do you need a debugger for a ten line string rversal func? Maybe not : but I'll tell you a little secret, I'd still use one to test it rigorously before handing it in to a system for an integration test. Total extra effort? About 5 minutes.

I'd write unit test. Also five minutes work and more likely to give proper coverage.

I realise that there is a core here who seem to think a debugger is almost evil : I sometimes question if they have used a real debugger on a real system in the real world where programmers are cycled on and off projects and people want to optimise their "up to speed" figures.

normally I don't get into what I work on, but yes I do work on projects

where people get cycled on and off. I'm one of the cycled. It's a real system. In the real world. Real people pay real money for the facilities it provides. And there is a requirement for more performance.

I'm not accusing you of it, but sometimes people seem to spend way too much time hunched over a debugger when a bit of thought might save them some time.

No doubt. And equally the opposite applies : in my experience more so.

I realise that according to some posters some projects get by without a debugger : I've never had the pleasure of such a project/system.

"A desk is a dangerous place from which to view the world" – LeCarre.

"there is nothing as practical as a good theory" Lewin

Re: Learning C