What type of programs are best written in C [closed]

Asked 15 years, 11 months ago Modified 2 years ago Viewed 42k times

26

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 years ago.

Improve this question

Joel and company expound on the virtues of learning C and how the best way to learn a language is to actually write programs using that use it. To that effect, which types of applications are most suitable to the C programming language?

Edit: I am looking for what C is good at. This likely does not coincide with the best way of learning C.

c

Share

edited Jan 13, 2009 at 23:58

Improve this question

Follow

community wiki
2 revs
dmanxiii

I wonder if people will answer this as if the question were C or C++ – Chris Noe Jan 12, 2009 at 16:34

1 So far, it seems that the distinction is being drawn. – T.E.D. Jan 12, 2009 at 19:26

Those that matter. :-) – Ciro Santilli OurBigBook.com May 3, 2016 at 21:09

31 Answers

Sorted by: Highest score (default)

1 2 Next

45

Code where you need absolute control over memory management. Code where you need to be utterly in control of speed versus memory trade-offs. Very low-level file manipulation (such as access to the raw devices).

Examples include OS kernel, and embedded apps.

In the late 1980s, I was head of the maintenance team on a C system that was more than a million lines of code. It did database access (Oracle), X Windows graphics, interprocess communications, all sorts of good stuff. It ran on VMS and several varieties of Unix. But if I were to

recreate that system today, I sure wouldn't use C, I'd use Java. Others would probably use C#.

Share Improve this answer Follow

edited Jan 12, 2009 at 16:33

community wiki
2 revs
Paul Tomblin

Even with C, you still have the OS in your way. If you take that away, the processor still rewrites your instructions, reorders reads, etc. Also, there is no reason to use C for manipulation of raw devices, unless you need to do a very simple transformation to terabytes of data. – jrockway Jan 12, 2009 at 16:17

@jrockway - When you say "OS in your way", I assume you are thinking of the process space, VM, protection rings kind of stuff. Understand that not all of us write for the *nix, Windows, etc. In the RTOS/embedded world, we still doing lots of stuff in C so the language itself doesn't get in our way! – Tall Jeff Jan 14, 2009 at 1:13

+1 memory mapped I/O would be nearly impossible without C. – yogman Jan 15, 2009 at 1:41

28

Low level functions such as OS kernel and drivers. For those, C is unbeatable.

Share Improve this answer Follow

edited Jan 12, 2009 at 16:16

community wiki
2 revs, 2 users 67%
Otávio Décio

While those are clearly the primary real world domains for C, they aren't really where you would want to start to learn the C language. – Steve Fallows Jan 12, 2009 at 15:58

2 I know, but as a good programmer I tend to answer what was asked. – Otávio Décio Jan 12, 2009 at 16:01

I don't think C is particularly good for writing drivers in. It just happens to be popular for that. – jrockway Jan 12, 2009 at 16:15

1 @Steve: agreed. Learning C *and* driver architecture at the same time isn't really good advice for the questioner to actually not get frustrated or completely confused. – ctacke Jan 12, 2009 at 16:16

True, trying to learn C and very low level systems at the same time is a Bad Idea. The question was, however, "what are good applications of the C language" not "where is a good place to start learning C". ocdecio answered the questions as it was asked. – Toji Jan 12, 2009 at 16:20

20

You can use C to write anything. It is a very general purpose language. After doing so for a little while you will understand why there are other "higher level" languages available.

"Learn C", by all means, but don't don't stick with it for too long. Other languages are far more productive.

I think the gist of the people who say you need to learn C is so that you understand the relationship between high level code and the machine internals and what exaclty happens with bits, bytes, program execution, etc.

Learn that, and then move on.

Share Improve this answer Follow

answered Jan 12, 2009 at 16:19

community wiki
Tim

But do C++ and Objective-C count as higher-level? I think that's debatable and there's still vast amounts of code created in them. – Cruachan Jan 12, 2009 at 16:28

well, they are certainly higher level than C. I did not specify which one(s) to use. There again it is a choice of appropriateness to the domain, the organization, etc. – Tim Jan 12, 2009 at 16:31

2 They're higher-level in the sense that they provide more means of abstraction. – Rob Jan 12, 2009 at 22:55

1 The main productivity boost is auto memory management. Therefore C++ and Objective-C do not qualify as higher-level in this case. – jfs Jan 14, 2009 at 1:22

1 Huh? that makes no sense – Tim Jan 14, 2009 at 3:56

Those 100 lines of python code that were accounting for 80% of your execution time.

15

Share Improve this answer

Follow

edited Nov 22, 2014 at 18:22

community wiki
2 revs, 2 users 67%
Jimmy

I would argue it is the algorithm you need to consider to optimize first. – JP Richardson Jan 12, 2009 at 16:23

11

Small apps that don't have a UI, especially when you're trying to learn.

Edit: After thinking a little more on this, I'd add the following: if you already know a higher-level language and you're trying to learn more about C, a good route may be to not create a whole new C app, but instead create a C DLL and add functions to it that you can call from the higher language. In this way you can replace simple functions that your high language already has to ensure that you C function does what it should (gives you pre-built testing), lets you code mostly in what you're familiar with, uses the language in a problem you're already familiar with, and teaches you about interop.

Share Improve this answer

Follow

edited Jan 12, 2009 at 16:14

community wiki

2 revs
ctacke

Why was this given a -1? – jmucchiello Jan 12, 2009 at 16:22

It was actually give a -3 so far. – ctacke Jan 12, 2009 at 17:15

It was voted down because many large applications that do have an UI are written in C: MS Word, MS Excel... – florin Jan 14, 2009 at 0:07

8

Anything where you think of using assembly.

Share Improve this answer Follow

answered Jan 12, 2009 at 16:02

community wiki
Michael Kohne

Most processors have instructions that accessible in plain C; bitscans are a nice example of that. – Jasper Bekkers Jan 16, 2009 at 18:01

7

- Number crunching (for example, libraries to be used at a higher level from some other language like Python).
- Embedded systems programming.

answered Jan 12, 2009 at 15:59

community wiki
Federico A. Ramponi

A lot of people are saying OS kernel and device drivers which are, of course, good applications for C. But C is also useful for writing any performance critical applications that need to use every bit of performance the hardware is capable of.

I'm thinking of applications like database management systems (mySQL, Oracle, SQL Server), web servers (apache, IIS), or even we browsers (look at the way chrome was written).

You can do so many optimizations in C that are just impossible in languages that run in virtual machines like Java or .NET. For instance, databases and servers support many simultaneous users and need to scale very well. A database may need to share data structures between multiple users (threads/processes), but do so in a way that efficiently uses CPU caches. In C, you can use an operating system call to determine the size of the cache, and then align a data structure appropriately to the cache line so that the line does not "ping pong" between caches when multiple threads access adjacent, but unrelated data (so called "false sharing). This is one example. There are many others.

Share Improve this answer

Follow

answered Jan 12, 2009 at 19:23

community wiki
RussellH

6

A bootloader. Some assembly also required, which is actually very nice..

Share Improve this answer

Follow

answered Jan 12, 2009 at 16:11

community wiki
sajith

5

1. Where you feel the need for 100% control over your program. This is often the case in lower layer OS stuff like device drivers, or real embedded devices based on MCU:s etc etc (all this and other is already mentioned above)
2. But please note that C is a mature language that has been around for many years and will be around for many more years, it has many really good debugging tools and still a huge number off developers that use it. (It probably has lost a lot to more trendy languages, but it is still huge) All its strengths and weaknesses are well know, the language will

probably not change any more. So there are not much room for surprises...

This also means that it would probably be a good choice if you have a application with a long expected life cycle.

/Johan

Share Improve this answer Follow

answered Jan 12, 2009 at 21:03

community wiki
Johan

2

Anything where you need a minimum of "magic" and need the computer to do **exactly** what you tell it to, no more and no less. Anything where you don't trust the "magic" of garbage collection to handle your memory because it might not be as efficient as what you can hand-code. Anything where you don't trust the "magic" of built-in arrays, strings, etc. not to waste too much space. Anything where you want to be able to reason about exactly what ASM instructions the compiler will emit for a given piece of code.

In other words, not too much in the real world. Most things would benefit more from higher level abstraction than from this kind of control. However, OS code, device drivers, and a few things that have to be near optimal in **both** space and speed might make sense to write in C.

Higher level languages can do pretty well competing with C on speed, but not necessarily on space.

Share Improve this answer Follow

answered Jan 12, 2009 at 16:27

community wiki
dsimcha

+1 for the first paragraph, -1 for the second. OS code, device drivers, and especially embedded devices are not "the real world"? – Jeanne Pindar Oct 25, 2009 at 22:59

OS code, device drivers and embedded devices *are* the real world, but a relatively small portion of it. Hence, "not too much" instead of "nothing". – dsimcha Oct 26, 2009 at 12:45

There are more embedded devices around than there are computers - especially since each computer contains several of them. :-) – Jeanne Pindar Oct 26, 2009 at 17:31

2

Embedded stuff, where memory-usage and cpu-speed matters.
The interrupt handler part of an OS (and maybe two or three more functions in it).

Even if some of you will now start to bash heavily on me now:
I dont think that any decent app should be written in C - it is way too error prone. (and yes, I do know what I am talking about, having written an awful lot of code in C

myself (OSes, compilers, VMs, network protocols, RT-control stuff etc.).

Using a high level language makes you so much more productive. Speed is typically gained by keeping the 10-90 rule in mind: 90% of CPU time is spent in 10% of your code (which you should optimize first). Also, using the right algorithm might give more performance than optimizing bits in C. And doing things right in C is so much more trouble.

PS: I do really mean what I wrote in the second sentence above; you can write a complete high performance OS in a high level language like Lisp or Smalltalk, with only a few minor parts in C. Think how the 70's Lisp machines would fly on todays hardware...

Share Improve this answer Follow

answered Jan 12, 2009 at 16:40

community wiki
blabla999

2

Garbage collectors!

Also, simply programs whose primary job is to make operating-system calls. For example, I need to write a short C program called `timeout` that

- Takes a command line as argument, with a number of seconds for that command to run

- Forks two child processes, one to run the command and one to sleep for N seconds
- When the first of the child processes exits, kills the other, then exits

The effect will be to run a command with a limit on wall-clock time.

I and others on this forum have tried several different solutions using shells and/or perl. All are convoluted and none quite do the right thing. In C the solution will be easy, because all the OS facilities are right where you can get at them.

Share Improve this answer Follow

answered Jan 13, 2009 at 5:58

community wiki
Norman Ramsey

1

A few kinds that I can think of:

- Systems programming that directly uses Unix/Linux or Win32 system calls
- Performance-critical code that doesn't have much string manipulation in it (e.g., number crunching)
- Embedded programming or other applications that are severely resource-constrained

Basically, C gives you portable, efficient access to the native capabilities of the machine; everything else is your responsibility. In particular, string manipulation in C is tedious, error-prone, and generally nasty; the most effective way to do extensive string operations with C may be to use it to implement a language that handles strings better...

Share Improve this answer Follow

answered Jan 12, 2009 at 19:31

community wiki
comingstorm

1

examples are: embedded apps, kernel code, drivers, raw sockets.

However if you find yourself more productive in C then go ahead and build whatever you wish. Use the language as a tool to get your problem solved.

Share Improve this answer Follow

answered Jan 12, 2009 at 22:45

community wiki
Signal9

---

c compiler

1

Share Improve this answer

Follow

answered Jan 14, 2009 at 14:13

community wiki
plan9assembler

1

Researches in maths and physics. There are probably two alternatives: C and C++, but such features of the latter as encapsulation and inheritance are not very useful there. One could prefer to use C++ "as a better C" or just stay with C.

Share Improve this answer

Follow

answered Jan 16, 2009 at 17:54

community wiki
Andrew T

1

Well most people are suggesting system programming related things like OS Kernels , Device Drivers etc. These are difficult and Time consuming. Maybe the most fun thing to with C is console programming. Have you heard of the HAM SDK? It is a complete software development kit for the Nintendo GBA , and making games for it is fun. There is also the CC65 Compiler which supports NES Programming (Althought Not Completely). You can also make good Emulators. Trust Me , C is pretty helpful. I was originally a Python fan, and hated C because it was

complex. But after yuoget used to it, you can do anything with C. Now I use CPython to embed Python in my C Programs(if needed) and code mostly in C. C is also great for portability , There is a C Compiler for like every OS and Almost Every Console And Mobile Device. Ive even seen one that supports some calculators!

Share Improve this answer Follow

answered Apr 9, 2011 at 13:46

community wiki
Burning Prodigy

1

Well, if you want to learn C and have some fun at the same time, might I suggest obtaining NXC and a Lego Mindstorms set? NXC is a C compiler for the Lego Mindstorms.

Another advantage of this approach is that you can compare the effort to program the Mindstorms "brick" with C and then try LEJOS and see how it goes with Java.

All great fun.

Share Improve this answer Follow

edited Dec 17, 2013 at 22:46

community wiki
2 revs, 2 users 80%
Huntrods

NXC isn't really C, though it is based on C - it's more like C with NXTensions (Oops, I may have coined a new phrase) and limitations. – Joe D Aug 26, 2010 at 18:08

1

Implicit in your question is the assumption that a 'high-level' language like Python or Perl (or Java or ...) is fast enough, small enough, ... enough for most applications. This is of course true for most apps and some choice X of language. Given that, your language of choice almost *certainly* has a foreign function interface (FFI). Since you're looking to *learn* C, create a module in the FFI built in C.

For example, let's assume that your tool of choice is Python. Reimplement a subset of Numpy in C. Since C is a pretty fast language, and has, in C99, a clear numeric library interface, you'll get the opportunity to experience the power of C in an appropriate setting.

Share Improve this answer Follow

edited Dec 9, 2022 at 19:03

community wiki
2 revs, 2 users 67%
Don Wakefield

ISAPI filters for Internet Information Server.

Share Improve this answer

answered Jan 12, 2009 at 16:09

0

Follow

community wiki
Robert S.

0

Before actually write C code, i would suggest first read good C code.
Choose subject you want to concentrate on, basically any application can be written in C, but i assume GUI application will be not your first choice, and find few open source projects to look into.

Not any open source project is best code to look. I assume that after you will select a subject there is a place for another question, ask for best open source project in the field.
Play with it, understand how it's working modify some functionality...
Best way to learn is learn from somebody good.

Share Improve this answer

Follow

answered Jan 12, 2009 at 16:22

community wiki
Ilya

0

Photoshop plugin filters. Lots and lots of interesting graphical manipulation you can do with those and you'll need pure C to do it in.

For instance that's a gateway to fractal generation, fourier transforms, fuzzy algorithms etc etc. Really anything that can manipulate image/color data and give interesting results

Share Improve this answer Follow

answered Jan 12, 2009 at 16:22

community wiki
Cruachan

> Why on earth do you think such a job has to be done in C? Lots of languages can compile to DLLs. Lots of languages are well suited to mathsy transforms (FP languages in particular). – slim Jan 12, 2009 at 16:48

> Well of course in theory you can write a Photoshop plugin in anything that can compile a dll. In practice if you're ever worked with the Photoshop plugin API - and I have - using anything else but C would display an alarming degree of masochism – Cruachan Jan 12, 2009 at 21:38

0

Don't treat C as a beefed up assembler. I've done some serious app's in it when it was the natural language (e.g., the target machine was a Solaris box with X11).

Write something with some meat on it. Write a client server chess program, where the AI is on a server and the UI is displaying in X11; once you've done that you will really know C.

Share Improve this answer Follow

edited Jan 12, 2009 at 18:53

community wiki
2 revs
CodeSlave

> Or you could do the same thing in Lisp or Perl, and have several thousand less lines of code, but with the same performance. (Waiting for the user to click a button isn't very CPU intensive.) – jrockway Jan 12, 2009 at 16:18

> I was sort of treating this question as a "what should I write to learn C" - which was wrong of me. However, imagine that that server program is handling games for hundreds or thousands of users. Or perhaps you are trying to beat the world chess champion. Now you are starting to need performance. – BIBD Jan 12, 2009 at 16:26

0

I wonder why nobody stated the obvious:

Web applications.

Share Improve this answer Follow

answered Jan 12, 2009 at 19:09

community wiki

Knuth

You mean the obviously wrong? – T.E.D. Jan 12, 2009 at 19:20

2 +1 for my kind of humour (I hope nobody takes it seriously though) – Tamas Czinege Jan 12, 2009 at 19:34

0

Any place where the underlying libraries are entirely in C is a good candidate for staying in C - openGL, Lua extensions, PHP extensions, old-school windows.h, etc.

Share Improve this answer Follow

answered Jan 12, 2009 at 21:13

community wiki
Nick Van Brunt

Well, if you ask me, even though the OpenGL API is C, anything that requires 3D graphics would probably better off with C++, to reduce complexity. – Tamas Czinege Jan 13, 2009 at 6:14

0

I prefer C for anything like parsing, code generation - anything that doesn't need a lot of data structure (read OOP). It's library footprint is very light, because class libraries are non-existent. I realize I'm unusual in this, but just as lots of things are "considered harmful", I try to have/use as little data structure as possible.

Share Improve this answer

Follow

answered Jan 12, 2009 at 22:54

community wiki
Mike Dunlavey

0

Following on from what someone else said. C seems a good language to implement the language in which you write the rest of your software.

And (mutatis mutandis) the virtual machine which runs the rest of your software.

Share Improve this answer

Follow

answered Jan 14, 2009 at 0:46

community wiki
interstar

0

I'd say that with the minuscule runtime environment and it's self-contained nature, you might start by creating some CLI utilities such as grep or tail (or half the commands in Unix). Anything that uses only STDOUT, STDIN and file manipulation is a good candidate.

This isn't exactly answering your question because I wouldn't actually CHOOSE to use C in such an app, but I hope it's answering the question you meant to ask--"what would be a good type of app to use learn C on?"

C isn't actually that bad a language--it's pretty easily to understand your code at an assembly language level which is quite useful, and the language constructs are few, leaving a very sparse language.

To answer your actual question, the very best use of C is the one for which it was created--porting itself (and UNIX) to other CPU architectures. This explains the lack of language features very well; it also explains the existence of Pointers which are really just a construct to make the compiler work less--any code created with pointers could be created without it (just as well optimized), but it becomes much harder to create a compiler to do so.

Share Improve this answer Follow

answered Jan 14, 2009 at 1:01

community wiki
Bill K

0

digital signal processing, all Pure Data extensions are written in C, this can be done in other languages also but has had good success in C

Share Improve this answer Follow

answered Dec 5, 2012 at 4:37

community wiki
aug2uag

1 2 Next